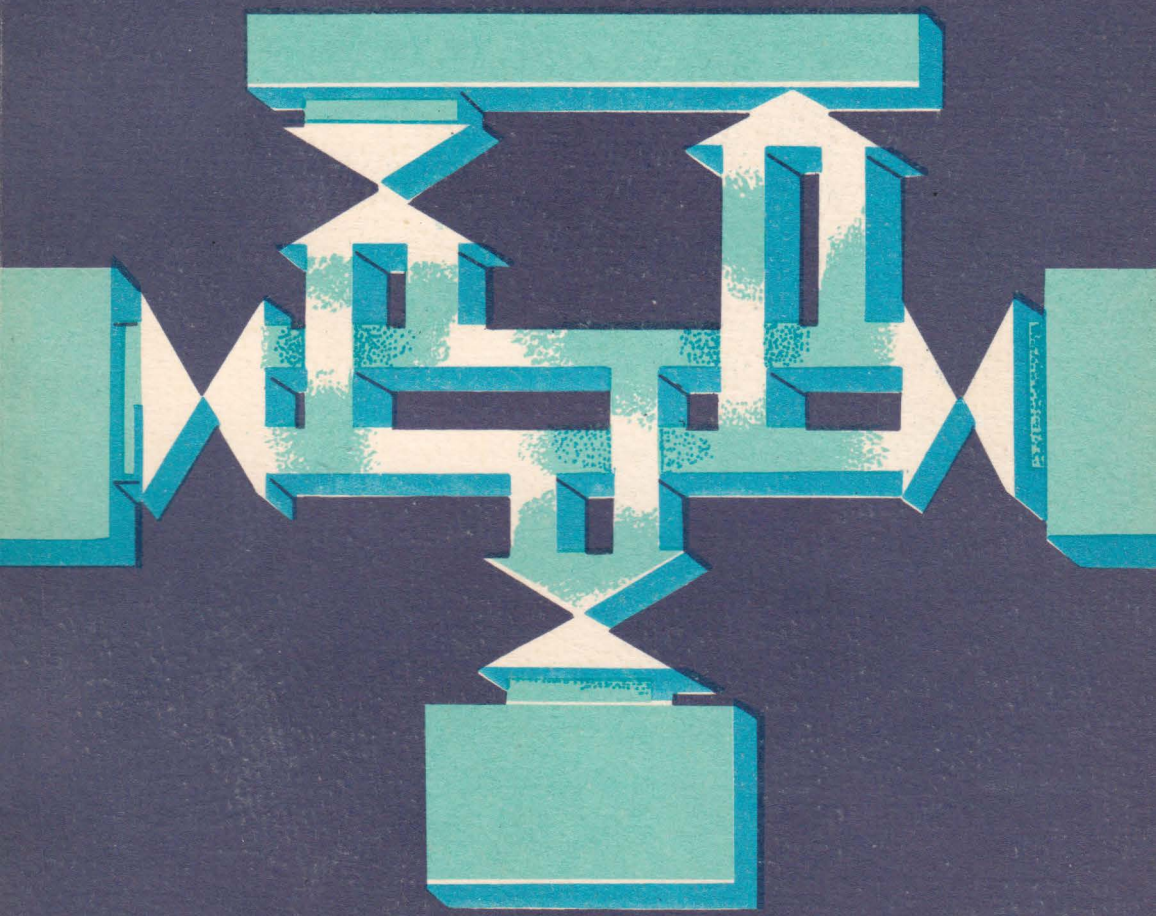
CRISTIAN LUPU
VLAD ŢEPELEA
EMIL PURICE

MICROPROCESOARE

APLICAȚII



EDITURA MILITARĂ

MICROPROCESOARE
APLICAȚII

Inginer CRISTIAN LUPU
Inginer VLAD ȚEPELEA
Inginer EMIL PURICE

MICROPROCESOARE

APLICAȚII

Cu o prefață de prof. dr. doc.
MIHAI DRĂGĂNESCU,
membru corespondent al Academiei Republicii Socialiste România

EDITURA MILITARĂ, BUCUREȘTI — 1982

Revizia și controlul științific au fost executate de:

Dr. ing. EMIL TUDOR

Dr. ing. ADRIAN DAVIDOVICIU

Redactor: colonel ing. ION MARINESCU
Coperta: AL. IULIAN
Tehnoredactor: GH. CUCOȘ
Desenele: GH. DUNĂREANU

*

Bun de tipar: 20.09.1982. Apărut: 1982.
Coli de tipar: 29,5.B113.

*

Tiparul executat sub comanda
nr. 1149 la
Întreprinderea poligrafică
„13 Decembrie 1918",
str. Grigore Alexandrescu nr. 89-97
București,
Republica Socialistă România

CUPRINS

Prefață IX
Introducere XI

1. CALCULATOARE, MICROPROCESOARE. ISTORIC ȘI PERSPECTIVE
1.1. Dezvoltarea calculatoarelor electronice în România 1
1.2. Circuite integrate. Microprocesoare 4
1.3. Evoluții arhitecturale. Perspective 10
Bibliografie 13

2. PROIECTAREA STRUCTURILOR DE CONTROL CU MICROPROCESOARE
2.1. Avantaje ale structurilor de control cu microprocesoare 15
2.2. Concepte care stau la baza proiectării structurilor de control cu microprocesoare 17
2.3. Etapele proiectării unei structuri de control cu microprocesor 21
Bibliografie 27

3. MICROPROCESOARE *SINGLE-CHIP*
3.1. Prezentare generală 29
3.2. Microprocesoare pe 8 biți 39
3.2.1. Microprocesorul 8080 39
3.2.2. Microprocesorul 8085A 42
3.2.3. Microprocesorul Z80 45
3.2.3.1. Generalități 45
3.2.3.2. Arhitectura internă 46
3.2.3.3. Descrierea semnalelor externe 48
3.2.3.4. Comportarea în întreruperi 50
3.2.3.5. Moduri de adresare 50
3.2.3.6. Setul de instrucțiuni 52
3.2.3.7. Structura unui microsistem realizat cu microprocesorul Z80 57
Bibliografie 58

4. SD-8080, UN SISTEM DE DEZVOLTARE PENTRU MICROPROCESOARE
4.1. Rolul sistemului de dezvoltare 60
4.2. Resurse hardware 63
4.2.1. Prezentare generală 63
4.2.2. Microprocesorul și circuitele aferente 64
4.2.2.1. Circuitul de ceas 64
4.2.2.2. Logica de stare și comandă a magistralelor 66
4.2.3. Memoria 79
4.2.3.1. Memoria PROM implementată cu circuite 3601 81
4.2.3.2. Memoria RAM implementată cu circuite 2102A 87
4.2.3.3. Memoria RAM realizată cu circuite F16K 92
4.2.3.4. Memoria REPROM realizată cu circuite 2708 101
4.2.4. Sistemul de întreruperi 103

4.2.5. Sistemul de I/E ... 107
4.2.5.1. Generalități ... 107
4.2.5.2. Repartizarea adreselor de I/E ... 108
4.3. Resurse software ... 112
4.3.1. Monitorul ... 113
4.3.2. Sistemul rutinelor de I/E ... 114
4.3.3. Programele de test ale memoriei ... 118
4.3.4. Limbajul BASIC ... 120
Bibliografie ... 123

5. LIMBAJUL DE ASAMBLARE 8080. TEHNICI DE PROGRAMARE

5.1. Generalități ... 124
5.2. Arhitectura microprocesorului 8080 ... 127
5.2.1. Moduri de adresare ... 129
5.2.2. Indicatorii de condiții ... 131
5.3. Setul de instrucțiuni 8080 ... 132
5.3.1. Formatul instrucțiunilor ... 132
5.3.2. Sintaxa instrucțiunilor ... 132
5.3.2.1. Cîmpul-etichetă ... 133
5.3.2.2. Cîmpul-cod ... 133
5.3.2.3. Cîmpul-operand ... 133
5.3.2.4. Cîmpul-comentariu ... 135
5.3.3. Reprezentarea internă a datelor ... 135
5.3.4. Descrierea instrucțiunilor 8080 ... 136
5.3.5. Pseudoinstrucțiuni ... 141
5.4. Tehnici de programare ... 145
5.4.1. Testarea indicatorilor de condiții ... 145
5.4.2. Testarea valorii anumitor biți ... 148
5.4.3. Testarea valorii unui octet ... 148
5.4.4. Testarea conținutului unui dublu cuvînt ... 149
5.4.5. Subrutine ... 151
5.4.6. Operații repetitive, bucle ... 154
5.4.6.1. Rutine de întîrziere (așteptare) ... 156
5.4.7. Tabele de salt ... 158
5.4.8. Tabele de date ... 161
5.4.9. Aritmetica în cod BCD ... 171
5.4.9.1. Reprezentarea numerelor ... 171
5.4.9.2. Conversia unui număr din cod binar în cod BCD și invers ... 175
5.4.9.3. Adunarea a 2 numere ... 188
5.4.9.4. Scăderea a 2 numere ... 190
5.4.9.5. Înmulțirea a 2 numere ... 192
5.4.9.6. Împărțirea a 2 numere ... 198
5.4.9.7. Compararea a 2 numere ... 206
Bibliografie ... 209

6. APLICAȚII ALE MICROPROCESORULUI 8080

6.1. Interfața pentru consolă-serie ... 210
6.2. Interfața pentru perforatorul de bandă ... 221
6.3. Interfața pentru cititorul de bandă perforată ... 225
6.4. Interfața pentru cititorul de cartele ... 228
6.5. Interfața pentru imprimanta paralelă ... 237
6.6. Rutina pentru calculul codului de control ciclic ... 244
6.7. Generatorul vitezelor de teletransmisie ... 251
6.8. Interfața pentru două unități de casete magnetice ... 255
6.8.1. Generalități ... 255
6.8.2. Descrierea unității de casete magnetice folosite ... 256
6.8.3. Descrierea părții hardware a interfeței ... 261
6.8.4. Rutinele de comandă ... 267

6.9. Folosirea microprocesorului 8080 în întreruperi 275
6.9.1. Generalități 275
6.9.2. Programarea întreruperilor în cadrul SD-8080 280
6.9.3. Circuitul specializat de tratare a întreruperilor 8259 284
Bibliografie 291

7. MICROPROGRAMARE

7.1. Conceptul tradițional de microprogramare 293
7.2. Conceptul modern de microprogramare 296
7.2.1. O structură de control microprogramată 296
7.2.2. Caracteristici ale structurilor de control microprogramate 300
7.2.2.1. Organizarea memoriei de microprograme 300
7.2.2.2. Formatul microinstrucțiunii 302
7.2.2.3. Implementarea microinstrucțiunii 305
7.3. Avantajele și dezavantajele microprogramării 307
7.3.1. Avantajele microprogramării 309
7.3.2. Dezavantajele microprogramării 312
7.4. Aplicații ale microprogramării 312
Bibliografie 316

8. SISTEME SUPORT PENTRU MICROPROGRAMARE

8.1. Utilitatea sistemelor suport pentru microprogramare 318
8.1.1. Funcțiile sistemelor suport pentru microprogramare 318
8.1.2. *System* 29 — un sistem suport pentru microprogramare 319
8.1.3. Un exemplu de utilizare a *System*-ului 29 323
8.2. SD-8080 folosit ca sistem suport pentru microprogramare 328
8.2.1. Memoria WCS pentru simularea memoriei de microprograme 328
8.2.2. Un microasamblor dedicat 330
8.3. Asambloare de microprograme 331
8.3.1. Generalități 331
8.3.2. Meta-asamblorul METASM 335
8.3.2.1. Caracteristici generale 335
8.3.2.2. Definirea limbajului de asamblare 337
8.3.2.3. Scrierea microprogramului 339
8.3.2.4. Facilități extinse ale meta-asamblorului METASM 340
Bibliografie 341

9. MICROPROCESOARE *BIT-SLICE*

9.1. Generalități 343
9.2. Familia Am 2900 344
9.2.1. Microprocesorul *bit-slice* Am 2901 344
9.2.2. Generatorul de transport anticipat Am 2902 350
9.2.3. Microprocesoarele *bit-slice* Am 2903/29203 351
9.2.4. Circuitul pentru controlul deplasării și al indicatorilor de condiții Am 2904 357
9.2.4.1. Structura circuitului 358
9.2.4.2. Instrucțiunile lui Am 2904 360
9.2.5. Secvențiatoarele de microprogram Am 2909/2911 și circuitul de control Am 29811 364
9.2.5.1. Structura secvențiatoarelor Am 2909/2911 364
9.2.5.2. Instrucțiunile de adresare ale secvențiatoarelor Am 2909/2911 și ale circuitului Am 29811 368
9.2.6. Controlorul de microprogram Am 2910 370
9.2.6.1. Structura circuitului 371
9.2.6.2. Instrucțiunile lui Am 2910 373
9.2.7. Controlorul de program Am 2930 379
9.2.7.1. Structura circuitului 379
9.2.7.2. Instrucțiunile de adresare ale controlorului Am 2930 381
9.3. Alte familii și circuite *bit-slice* 385
9.3.1. Seria Intel 3000 385
9.3.1.1. Controlorul de microprogram Intel 3001 385

PREFAȚĂ

Apariția microprocesorului în anul 1971 constituie unul din marile evenimente ale tehnologiei contemporane. El a fost rezultatul unei evoluții tehnologice rapide în domeniul microelectronicii, revoluție care debutase în anul 1958 prin apariția circuitului integrat. Ideea circuitului integrat fusese formulată din anul 1952 și realizată experimental șase ani mai tîrziu. Ideea microprocesorului a fost prezentată în 1969 și împlinită experimental în numai doi ani.

Microprocesorul determină o revoluție tehnică, revoluția microprocesorului sau, sub o denumire mai generală, revoluția microelectronică. Această revoluție antrenează toate domeniile tehnologiei; nu există domeniu de activitate umană în care microprocesorul să nu poată fi implicat. Microprocesorul este unul din uneltele istorice ale omenirii. După uneltele de piatră, de bronz și fier, după mașină, apare microprocesorul din siliciu ca o unealtă informațională care preia sub o formă abstractă, simbolică, funcții în raport cu realitatea tehnică, economică, culturală, socială. El poate conduce procese, mașini, roboți, poate înmagazina inteligență, poate înlocui părți mecanice și electromecanice ale multor utilaje.

Pentru a fi utilizat microprocesorul trebuie să fie programat. De la bun început el a făcut să fie ștearsă frontiera dintre electronică și informatică. De la bun început el s-a dovedit a fi primul dispozitiv electronic funcțional în sensul nou al unei electronici funcționale îndreptate către funcții în raport cu uneltele de producție, cu omul, cu societatea și cu materia pe care cercetătorul caută s-o aprofundeze. Datorită lui electronica funcțională este și o informatică funcțională. Datorită lui este posibilă informatica distribuită și structurată pe niveluri ierarhice.

Dar microprocesorul nu produce numai o revoluție tehnică ci determină, însoțit de programul informatic, în special cel cu inteligență artificială, prin cuplare cu vechea mecanică, o nouă revoluție industrială, adică un proces care va antrena mari schimbări în viața economică și socială. Fenomenul este studiat cu atenție în întreaga lume. Pentru că microprocesorul asigură suportul material al acestei

revoluții, el este considerat primul factor determinant. Clubul de la Roma a elaborat recent un nou raport asupra acestor procese cu mari implicații asupra societății intitulat semnificativ „Microelectronica și societatea" (Pergamon Press, Oxford, 1982), atrăgînd atenția asupra necesității unei conștientizări a omenirii în vederea utilizării acestei tehnologii pentru mai bine și nu pentru mai rău. În același timp, revoluția microelectronică este apreciată ca ireversibilă. Preocupări similare, în cadrul Comisiei pentru revoluția științifică și tehnică a Academiei Republicii Socialiste România, mi-au permis să mă refer într-o serie de lucrări asupra acestor probleme, din anul 1976 pînă în prezent, multe puncte de vedere coincizînd cu cele ale studiului Clubului de la Roma.

Cum trebuie însă să înceapă revoluția microprocesorului? Prin studiul unor cărți ca aceasta de față scrisă de ing. Cristian Lupu, ing. Vlad Țepelea și ing. Emil Purice, simultan cu lucrul efectiv cu un microprocesor, dacă se poate în scopul unei aplicații concrete.

Volumul pe care îl prezentăm cititorului este realizat de un grup de autori care lucrează într-un institut de cercetare din domeniul calculatoarelor electronice și este orientat către aplicații. Din acest punct de vedere se deosebește de celelalte volume referitoare la microprocesoare apărute la noi. Competența autorilor și caracterul lucrării fac ca acest volum să fie o reală contribuție la extinderea utilizării microprocesoarelor în economia națională în conformitate cu Programul-Directivă de cercetare științifică, dezvoltare tehnologică și de introducere a progresului tehnic în perioada 1981—1990 și direcțiile principale pînă în anul 2000, aprobat de Congresul a XII-lea al Partidului Comunist Român.

MIHAI DRĂGĂNESCU

membru corespondent
al Academiei Republicii Socialiste România

30 mai 1982

Microprocesoarele — exponente de vîrf ale tehnologiei electronice actuale — constituie elemente de prelucrare a informației realizate sub forma unor circuite integrate pe scară largă ce pot îngloba într-o singură pastilă de siliciu, într-un singur chip, *de la cîteva mii pînă la sute de mii de dispozitive de tip tranzistor. Adevărate calculatoare în miniatură utilizabile de orice proiectant de structuri de control* electronice, microprocesoarele permit un acces de nivel superior la puterea de calcul, o „distribuire informatică" remarcabilă.*

Apariția și extinderea utilizării acestor circuite integrate au cunoscut o dezvoltare impetuoasă. În răstimpul a două decenii s-a ajuns de la tranzistor, realizat în anul 1948, la circuite integrate, fabricate în anul 1958, și la prima familie de microprocesoare pe 4 biți, elaborată în anul 1971. Saltul calitativ de la tuburile electronice și tranzistoare la circuite integrate, iar în prezent la microprocesoare a constituit începutul celei de-a doua revoluții industriale. Spre deosebire de prima revoluție industrială — la baza căreia a stat mecanica, mașinismul —, a doua revoluție industrială are ca fundament microelectronica, automatica, calculatoarele, cibernetica aplicate extensiv, cu mari implicații în toate domeniile de activitate ale societății moderne.

Pentru a ilustra măsura în care automatizarea a revoluționat tehnica actuală, considerăm că este suficient să amintim că prin reproiectarea unui aparat telex și încorporarea doar a unui singur microprocesor s-au eliminat 936 de piese mecanice, iar durata medie de fabricație s-a redus de la 75,3 la 17,7 ore.

* Termenul de control, folosit frecvent în lucrare, are un sens aparte și desemnează o noțiune ce înglobează diverse procese de comandă, conducere, dirijare și reglare.

În concepția Partidului Comunist Român privind dezvoltarea forțelor de producție și așezarea întregii activități industriale pe baza celor mai noi cuceriri ale științei și tehnicii, un rol important îl ocupă cercetarea științifică, crearea de noi mijloace tehnice în măsură să revoluționeze laturi importante ale activității umane.

Dînd expresie rolului pe care îl are automatizarea în ansamblul procesului revoluționar de dezvoltare a forțelor de producție, Conferința Națională a Partidului Comunist Român din anul 1967 a aprobat măsurile de perfecționare a conducerii și planificării economiei naționale cu echipamente moderne de automatizare și de calcul. În perioada care a urmat Congreselor al X-lea și al XI-lea ale partidului, ținînd seama de realizările și experiența dobîndită, au fost adoptate noi măsuri pe linia creării sistemului național de prelucrare a datelor, promovării cibernetici și informaticii în producție și gestiune.

Relevînd însemnătatea acestui important domeniu tehnico-științific cu mari implicații în dezvoltarea economiei naționale, tovarășul Nicolae Ceaușescu arăta la cel de-al XII-lea Congres al partidului: Un rol însemnat vor avea în acest scop extinderea mecanizării și automatizării, realizarea de linii complet automatizate ..., folosirea mașinilor-agregat multifuncționale, a roboților industriali și microprocesoarelor. Va trebui acționat pentru conducerea cibernetizată a proceselor tehnologice ..., folosirea tehnicii electronice în programarea producției, în lucrările de calcul și evidență și, pe această bază, reducerea la strictul necesar a personalului funcționăresc.*

Acționîndu-se în direcția înfăptuirii acestor obiective s-au obținut noi succese pe linia dezvoltării în ritm susținut a industriei producătoare de mijloace de prelucrare, de transmitere și de afișare a datelor, au luat ființă numeroase centre de calcul la întreprinderi, în cadrul centralelor și combinatelor industriale, al ministerelor și instituțiilor de cercetare și învățămînt; s-a realizat o puternică rețea de centre și oficii de calcul în profil teritorial pentru fiecare județ al țării; automatizarea proceselor de producție a cunoscut o largă dezvoltare.

În Programul-Directivă de cercetare științifică, dezvoltare tehnologică și de introducere a progresului tehnic în perioada 1981—1990

---

* NICOLAE CEAUȘESCU, *Raport la cel de-al XII-lea Congres al Partidului Comunist Român*, Editura politică, București, 1979, p. 41.

și direcțiile principale pînă în anul 2000 se arată că: În electronică și electrotehnică, cercetarea științifică se va concentra asupra realizării de componente și produse cu performanțe ridicate care asigură promovarea largă, în toate ramurile economiei naționale, a automatizării complexe și cibernetizării, inclusiv prin utilizarea roboților industriali.

În acest scop, se va acorda o atenție deosebită producerii de componente discrete de puteri și frecvențe ridicate, circuite integrate speciale, microprocesoare, memorii, componente optoelectronice. Se va pune, de asemenea, accentul pe realizarea de traductoare, serii de motoare specializate, echipamente de măsură și control, elemente electromecanice, componente specializate, în vederea extinderii elementelor automatizate.

Vor fi elaborate noi tipuri de microcalculatoare, echipamente de colectare și introducere a datelor, programe de bază pentru micro și minicalculatoare, calculatoare de proces.*

Lucrarea de față se înscrie în literatura de profil ca o expresie a triadei învățămînt-cercetare-producție, atît de importantă pentru implementarea acestei tehnici de vîrf reprezentată de microprocesoare. Ea se adresează în egală măsură atît studenților facultăților de electronică și automatică, cît și inginerilor chemați să proiecteze, să realizeze și să utilizeze structuri de control cu microprocesoare destinate celor mai variate procese de automatizare.

Autorii aduc mulțumiri tovarășului prof. dr. doc. ing. Mihai Drăgănescu, membru corespondent al Academiei R.S.R., director general al Institutului central pentru Conducere și Informatică, precum și tovarășilor dr. ing. Emil Tudor, directorul Centrului de Cercetări pentru Tehnică de Calcul și dr. ing. Adrian Davidoviciu, director adjunct al Institutului central pentru Conducere și Informatică, pentru îndrumarea și sugestiile făcute, care au condus la completarea și îmbunătățirea conținutului lucrării. Autorii sînt recunoscători, totodată, Editurii militare pentru solicitudinea și receptivitatea pe care le manifestă consecvent în promovarea lucrărilor tehnico-științifice ce vizează introducerea progresului tehnic, precum și pentru modul în care a reușit să transforme într-o activitate de plăcută colaborare dificila muncă de metamorfozare a manuscrisului brut într-o carte tipărită.

---

* *Programul-Directivă de cercetare științifică, dezvoltare tehnologică și de introducere a progresului tehnic în perioada 1981—1990 și direcțiile principale pînă în anul 2000*, Editura politică, București, 1979, p. 19.

CALCULATOARE, MICROPROCESOARE. ISTORIC ŞI PERSPECTIVE

1.1. DEZVOLTAREA CALCULATOARELOR ELECTRONICE ÎN ROMÂNIA

Activităţile privind proiectarea şi realizarea calculatoarelor electronice în România datează din 1953. În acel an un colectiv condus de *Victor Toma* din cadrul Institutului de fizică al Academiei, mai tîrziu Institutul de fizică atomică, începe să se preocupe de unele aspecte ale tehnicii numerice: realizarea numărătoarelor electronice, studiul elementelor logice ale calculatoarelor, analiza algoritmilor, studiul dispozitivelor de memorare etc. [1]. După aceste lucrări de acomodare cu problematica s-a trecut la elaborarea unei instalaţii pilot de dimensiuni reduse. Această realizare va încuraja colectivul de la IFA ca, pe baza rezultatelor obţinute, să înceapă în *1955* proiectarea şi construirea primului calculator electronic românesc *CIFA-1, calculatorul Institutului de fizică al Academiei.*

În cadrul IFA s-a ales de la început linia de dezvoltare a calculatoarelor de tip numeric avîndu-se în vedere precizia de calcul ridicată şi faptul că proprietăţile de universalitate ale acestor maşini permiteau utilizarea lor în cele mai diverse domenii ale ştiinţei şi tehnicii [2].

În noiembrie 1955 calculatorul CIFA-1 va fi prezentat în cadrul Colocviului internaţional de matematică de la Dresda [3]. În *aprilie 1957* el va fi pus în funcţiune la IFA-Bucureşti.

Realizat cu cca 1 500 de tuburi electronice, făcînd parte deci din prima generaţie de calculatoare, CIFA-1 era o maşină universală de tip paralel, ce lucra în sistem binar. Numerele se reprezentau în virgulă fixă, fiind compuse din semn şi 30 de cifre binare. Calculatorul executa 21 de instrucţiuni de tipul cu o singură adresă (Tabelul 1.1) alcătuite dintr-un cod-operaţie de 5 biţi şi o adresă de 10 biţi. Deci un cuvînt reprezenta fie un număr în virgulă fixă, fie două instrucţiuni executabile succesiv de către calculator.

CIFA-1 era realizat din mai multe panouri montate în 3 *rack*-uri [3]. Memoria calculatorului, implementată pe un tambur magnetic, avea o capacitate de 1 024 de cuvinte a cîte 31 de biţi. Viteza motorului de 3000 rot/min a condus la un timp de acces mediu de 10 ms. Ca dispozitive de I/E*

* Intrare/Ieşire.

Tabelul 1.1. **Instrucțiunile calculatorului CIFA-1**

Instrucțiunea (în octal)	Semnificația operației
00	Stop.
01	Șterge A și B, adună M(x) cu conținutul registrului B, rezultatul în B.
02	Șterge A, adună M(x) cu conținutul registrului B, rezultatul în B.
03	Șterge A, adună valoarea absolută a locației de memorie M(x) cu conținutul registrului B, rezultatul în B.
04	Șterge A, citește M(x) în A.
05	Șterge B, transferă A în B, A rămîne neschimbat.
06	Șterge B, introduce complementul lui A în B, A rămîne nemodificat.
07	Șterge A, scade M(x) din B, rezultatul în B.
10	Introduce A în M(x).
11	Șterge A, introduce B în M(x), B rămîne neschimbat.
12	Șterge A, înmulțește M(x) cu B, rezultatul în B.
13	Șterge A, împarte B cu M(x), rezultatul în B.
14	Înmulțește B cu 2^x, rezultatul în B.
15	Împarte B cu 2^x, rezultatul în B.
16	Șterge A, citește M(x) în A, stop.
17	Șterge A, tipărește conținutul memoriei începînd cu locația M(x), stop.
20	Salt necondiționat la adresa x.
21	Salt condiționat pe rezultat negativ: dacă semnul lui B este 1 salt la adresa x, dacă semnul lui B este 0 programul continuă la adresa următoare.
22	Salt condiționat pe rezultat pozitiv: dacă semnul registrului B este 0 salt la adresa x, dacă semnul lui B este 1 programul continuă la adresa următoare.
23	Salt la instrucțiunea următoare.
24	*Breakpoint*, cu ajutorul unui comutator poate deveni salt la instrucțiunea următoare.

CIFA-1 avea prevăzute un cititor de bandă perforată și o mașină de scris electrică.

Calculatorul executa patru operații aritmetice: adunare, scădere, înmulțire și împărțire.

Timpul necesar execuției operațiilor aritmetice, incluzînd aici și unele operații de control specifice calculatorului, era de 150 μs pentru adunare sau scădere și de *5 ms* pentru înmulțire sau împărțire.

Calculatorul numeric CIFA-1 era o mașină asincronă, fiecărei operații afectîndu-i-se timpul necesar efectuării ei, terminarea fiind marcată de un impuls de sfîrșit de ordin care declanșa începutul operației următoare.

Asimilarea funcționării unui calculator electronic asincron cu un generator controlat de impulsuri va face obiectul tezei de doctorat susținută de Victor Toma în anul 1973 [4]. În aceeași lucrare, autorul introducea un concept nou în electronică. Este vorba despre conceptul de *armare* definit și explicat în [5] încă din anul 1961. Acest concept va fi reluat în literatura tehnică universală, cu același termen, abia opt ani mai tîrziu, în 1969. Importanța deosebită a conceptului consta în faptul că întreruperea generării impulsurilor, deși comandată aleator, se executa abia după terminarea completă a ciclului în curs de execuție.

În anul *1961* se realizează la Institutul Politehnic Timișoara calculatorul electronic cu tuburi *MECIPT-1* apoi, în *1965*, calculatorul cu tranzistori *MECIPT-2*. Avînd numeroase îmbunătățiri conceptuale și tehnologice aduse de un colectiv din care făceau parte *Vasile Baltac*, *Ion Mihăescu* și *Viorel Vițan*, MECIPT-2. era echipat cu memorie pe ferite construită în țara noastră. Tot la Timișoara, *Alexandru Rogojan* a elaborat o metodologie de proiectare a calculatoarelor numerice și a construit calculatorul *CETA* [6].

Din prima generație de calculatoare electronice construite la IFA, echipate cu tuburi electronice și memorie pe tambur magnetic, au fost realizate și puse în funcțiune, după CIFA-1, calculatoarele de tip paralel *CIFA-2* (*1959*), *CIFA-3* (*1961*), *CIFA-4* (*1962*), precum și calculatoarele de tip serie *CIFA-101* (*1962*) și *CIFA-102* (*1964*).

Calculatoare numerice din primele două generații s-au mai elaborat la Institutul de calcul din Cluj, de un colectiv condus de *Gheorghe Farcaș*, seria *DACCIC*, și în cadrul Ministerului Apărării Naționale de colective conduse de *Florin Munteanu*, *Emil Tudor*, *Anton Dogaru*, *Ioan Stroe*.

Calculatoare electronice analogice au fost proiectate și realizate la Academia Tehnică Militară, calculatorul *CAU-1*, precum și la Institutul de energetică al Academiei R.S.R., mașinile analogice *MECAN-1* și *MECAN-2*. Menționăm, de asemenea, simulatorul reactorului nuclear realizat la IFA-București și calculatorul analogic construit la IFA-Cluj.

În 1962, în cadrul laboratorului de calculatoare din IFA-București, va începe studiul circuitelor cu tranzistori în vederea folosirii lor în calculatoare [7]. Pe această bază au fost realizate calculatoarele electronice tranzistorizate *CET-500* în *1964* și *CET-501* în *1966*. Calculatoarele respective, din generația a doua, aveau memorie cu ferite.

După intrarea în funcțiune în anii 1970—1971 a Întreprinderii de calculatoare electronice, în țara noastră începe fabricarea calculatoarelor din generația a treia, *FELIX C-256*, pe baza licenței *IRIS-50*. În acest fel, în conformitate cu politica statului nostru privind dezvoltarea informaticii, începînd din anul 1971 România devine țară producătoare de calculatoare electronice pe cale industrială [8].

În 1967 ia ființă Institutul de Tehnică de Calcul, ITC, cu filiale la Timișoara și Cluj. Acest institut va elabora calculatoarele *FELIX C-32*, *FELIX C-512*, *FELIX C-1024*, *FELIX C-8010*, mașinile electronice de facturat și contabilizat *FC-15*, *FC-16*, *FC-30*, *FC-64*, *FC-128*, sisteme de colectare și introducere a datelor, seria de minicalculatoare *I-100*. Pe baza lui FELIX C-32 specialiștii din Institutul de cercetări, proiectări și automatizări, IPA, vor realiza primul calculator de proces *FELIX C-32 P* a cărui producție de serie începe în 1976.

Specialiștii din Institutul Politehnic București, în colaborare cu Întreprinderea de calculatoare electronice, realizează în *1978* sistemul *FELIX M-8* construit cu microprocesorul 8008, iar în *1979* microcalculatorul *FELIX M-18* construit cu microprocesorul 8080. Ei vor realiza, de asemenea, minicalculatoarele *CORAL 4001/4011* stîrnind un mare interes și pe piața externă.

Pe baza microprocesorului 8080 au mai fost realizate și introduse în producție de serie următoarele echipamente: microcalculatorul *FELIX M-118*

(ICE), microcalculatoarele de proces *ECAROM* 800/880 (IPA+FEA) și *SPOT-80* (ICE), automatele programabile *AP-101/117* (Automatica).

În anul 1975 intră în funcțiune Întreprinderea de echipamente periferice, IEPER, destinată dezvoltării producției de echipamente periferice. În același an ia ființă societatea mixtă Romcontrol Data, RCD, destinată de asemenea fabricării unor diverse dispozitive de I/E, avînd și rolul accelerării introducerii în România, în producție de serie, a unor echipamente perfecționate.

Cercetările în domeniul programării se efectuează în principal în cele trei institute de profil: ITC, ICI, Institutul central pentru conducere și informatică, și IPA. ITC se ocupă îndeosebi de elaborarea programelor de bază: software de operare, compilatoare, programe utilitare necesare exploatării etc. ICI are sarcina elaborării de programe și produse-programe aplicative și utilitare de uz general, iar IPA, de programe din domeniul aplicațiilor industriale.

Principalele sisteme de operare pentru microcalculatoarele românești sînt MON-18 și SFDX-18, elaborate de ICE, și RTOS-80 elaborate de ICI. Trebuie să menționăm, de asemenea, că diverse unități din economie au elaborat numeroase programe aplicative din care o bună parte sînt accesibile prin Biblioteca Națională de Programe (ICI).

1.2. CIRCUITE INTEGRATE. MICROPROCESOARE

Se poate considera că era electronicii începe în *1883* prin descoperirea *efectului Edison*. Atunci, cunoscutul inventator american *Thomas Alva Edison*, în încercările sale de a prelungi viața becurilor, a introdus în balonul vidat, care conținea filamentul incandescent din carbon, un electrod de metal. Edison a descoperit că dacă se aplică o tensiune pozitivă pe acel electrod, între filament și electrod va trece un curent electric. Acest fenomen, efectul Edison, va sta la baza tuturor tuburilor electronice și deci a întregii electronici, pînă la descoperirea tranzistorului.

În *1904 John Ambrose Flemming* descoperă *dioda cu vid*. După cum se știe, aceasta redresa undele de frecvență radio, dar nu le putea amplifica. La *25 octombrie 1906 Lee de Forest* înscrie patentul primei *triode*, faimosul tub cu trei elemente numit *audion*. Acest tub, cu o grilă de comandă între filament și anod, amplifica totuși foarte puțin. De abia după descoperirea reacției pozitive, a *regenerării*, cum s-a numit la început, de către *Edwin Howard Armstrong* se va putea vorbi despre amplificatorul cu tuburi electronice. Primul circuit regenerativ al lui Armstrong a funcționat la *22 septembrie 1912*, fiind repede produs pe scară industrială.

Termenul *electronică* va apărea prima oară în *1904* în titlul unei reviste: Jahrbuch der Radioaktivität und *Elektronik* [9]. Denumirea era utilizată pe atunci în studiile legate de mișcarea în diverse medii a particulelor încărcate electric.

Fabricarea tuburilor cu vid va fi puternic influențată de dezvoltarea tehnicii radio. În anii *1930* englezul *H.J. Round* realizează *tetroda* a cărei idee o avuseseră germanul *Walter Schottky* în *1919* și, independent, americanul *A.W. Hull* în *1923*. În *1929* olandezii *G. Holst* și *Benjamin D.H. Tellegen* inventează *pentodele* de mică putere de radio frecvență, în *1932* se realizează *heptoda*, iar în *1933*, *hexoda*. Dezvoltarea și îmbunătățirea tehnologiei tuburilor electronice sînt într-adevăr uimitoare: în 1932 revista *Electronics* publica o listă cu 300 de tuburi diferite [10].

Descoperirea tranzistorului în Laboratoarele Bell Telephone, de către *Walter Brattain*, *William Shockley* și *John Bardeen*, trece aproape neobservată. La *1 iulie 1948* ziarul *New York Times* anunța în cîteva rînduri, la rubrica *Noutăți Radio*, inventarea acestui dispozitiv ce va revoluționa într-adevăr lumea.

În domeniul tehnologiei electronice evoluția este, după cum se cunoaște, foarte rapidă. În *1958* apar primele dispozitive realizate prin difuzie — tranzistoarele *mesa*. Tot atunci *Fairchild Corporation* își va face marcată prezența în industria de tranzistori cu *procesul planar*. Deși se baza, ca și procesul *mesa*, pe difuzii și mascări, procesul planar avea cîteva îmbunătățiri. Tranzistorul *mesa* era relativ fragil și sensibil la efectul de contaminare în suprafață. Acest dezavantaj se înlătură în procesul planar, pasivînd suprafața semiconductorului, prin doparea oxidului de siliciu cu anumite impurități. De asemenea, procesul planar permitea difuzia bazei în colector, ceea ce va conduce la obținerea unei structuri mai puțin delicate. Dezavantajul procesului planar era că el nu asigura producerea de tranzistori de putere datorită rezistenței mari a materialului din care era făcut colectorul.

Problema va fi rezolvată în *1960* cînd Laboratorul Bell Telephone anunță metoda *epitaxială* de fabricare a tranzistorilor: creșterea unui strat subțire de siliciu pe un substrat de cristal. Această metodă permitea realizarea pe un substrat gros a unor tranzistori cu baza subțire, pentru frecvențe înalte și cu rezistivitate a colectorului mică, pentru puteri mari.

O altă problemă pe care și-o puneau în acea vreme fabricanții de semiconductoare era obținerea de circuite realizate într-un singur bloc semiconductor, a *circuitelor integrate*. În *februarie 1960* firma Fairchild va anunța familia de circuite integrate *MICROLOGIC*, iar în *martie* firma Texas Instruments va face cunoscut primul său circuit integrat, un circuit la comandă fabricat pentru armată. *Jack Kilby* de la Texas Instruments și *Robert Noyce*, pe atunci la Fairchild Corporation, vor fi creditați cu inventarea independentă a circuitului integrat.

La sfîrșitul anilor 1960 *Robert Noyce* și *Gordon Moore* pleacă de la Fairchild pentru a pune bazele firmei *Intel*. În octombrie 1969 Busicom Corporation din Japonia încheie un contract cu Intel pentru fabricarea unui set de circuite destinat realizării unei familii de calculatoare. Ca urmare a acestui contract în *iunie 1971* Intel va anunța familia de microprocesoare *4004*. Setul 4004, proiectat de *Federico Faggin*, acum președintele firmei *Zilog*, cuprindea o memorie fixă de 256 octeți, o memorie cu conținut aleator de 32 de biți cu un *port* de ieșire de 4 biți, un registru de deplasare de 10 biți și un microprocesor pe 4 biți.

În timp ce se lucra la 4004, Intel va fi contactată de compania Computer Terminal Corporation, acum Datapoint Corporation, în vederea realizării în tehnologia circuitelor integrate a registrelor și stivei destinate unui terminal inteligent. Intel va propune realizarea unui procesor într-un singur *chip*. Întîlnind însă un aspect arhitectural ce părea prea restrictiv, proiectanții de la Intel vor schimba structura *chip*-ului. Circuitul integrat rezultat nu va mai fi compatibil cu arhitectura mașinii proiectate de Computer Terminal și el nu va fi niciodată fabricat pentru destinația inițială: în schimb Intel va lansa în *1972* microprocesorul *8008*. Acesta a fost primul microprocesor de tip *single-chip** pe 8 biți. Circuitul realizat într-o capsulă de 18 pini conținea o unitate aritmetică, șapte registre, o stivă, și putea executa 45 de instrucțiuni.

În *1973* National Semiconductor va produce un circuit pe 4 biți, *General-Purpose Controller/Processor*, ce putea fi conectat într-o arhitectură de tip *bit-slice** pentru cuvinte de maximum 32 biți. Acest microprocesor era controlat de o memorie fixă de 100 microinstrucțiuni de cîte 23 de biți. Un alt procesor microprogramabil pe 4 biți, *Paralell Processor*, era fabricat, tot în acea vreme, de Rockwell. Microprocesorul avea un ciclu de ceas de 5 μs și putea executa 50 de instrucțiuni.

Microprocesoarele amintite, fabricate în tehnologie PMOS**, sînt considerate ca făcînd parte din prima generație. A doua generație, realizată în tehnologie NMOS***, va debuta în 1974 cu *Intel 8080* — versiunea îmbunătățită a lui 8008. Microprocesorul 8080 va deveni unul din cele mai cunoscute și întrebuințate circuite. Proiectat de *Masatoshi Shima*, care mai tîrziu va pleca la Zilog pentru a proiecta *Z80*, microprocesorul 8080 este compatibil cu 8008, fiind de zece ori mai rapid.

Primul compilator rezident pentru un limbaj de nivel înalt destinat microprocesoarelor a fost livrat în *1974* de Intel. Limbajul se numea *PL/M*.

Tot Intel va produce *8086*, primul microprocesor pe 16 biți de înaltă performanță, reprezentant al celei de-a treia generații de procesoare integrate. 8086, un circuit realizat în tehnologie HMOS**** ce conține 29 000 de tranzistori, va fi urmat de microprocesoarele pe 16 biți de la firmele Zilog (*Z8000*), Motorola (*M68000*) și National (*16000*). Memoria dinamică de 64 Kbiți va deschide era integrării pe scară foarte largă — VLSI*****, sute de mii de dispozitive de tip tranzistor în *chip*-uri de 1/4 in² cu linii de 1 μm. Pentru anii următori industria de semiconductoare are într-adevăr o perspectivă deosebită, cu totul alta decît aceea din vremea descoperirii modestului dispozitiv numit tranzistor, anunțat în treacăt de ziarul *New York Times* la 1 iulie 1948.

---

* Microprocesoarele actuale se pot clasifica, printre altele, în funcție de tehnologia în care sînt realizate, de tip MOS sau bipolară, în microprocesoare realizate într-o singură capsulă, *single-chip*, și microprocesoare *bit-slice* realizate în mai multe capsule, ce se pot conecta în cascadă.

** P-channel Metal Oxide Semiconductor

*** N-channel Metal Oxide Semiconductor

**** High performance MOS

***** Very Large Scale Integration.

Se pare că evoluţia hardware din anii 1970 va fi înlocuită în anii ce urmează de o evoluţie software [11]. Circuitele integrate prevăzute cu software nu vor fi de la început deosebit de eficiente în orice tip de aplicaţie. Dar fără acest efort iniţial făcut de fabricanţii de microprocesoare costul aplicaţiilor poate deveni prohibitiv datorită *crizei software* resimţită în prezent.

Intel a început procesul de introducere a soft-ului în hard odată cu lansarea microprocesorului 8086 şi a unor coprocesoare auxiliare. Pînă acum Intel a produs două astfel de coprocesoare: 8089, un procesor de intrare/ieşire, şi 8087, un procesor în virgulă mobilă. Datorită seturilor de instrucţiuni specializate aceste coprocesoare eliberează utilizatorul de sarcina de a scrie rutine software pentru I/E şi aritmetică în virgulă mobilă.

Andrew S. Grove, preşedintele firmei Intel, declara că după integrarea unităţilor centrale făcută în anii 1970, următoarea etapă va fi integrarea opţiunilor de performanţă, etapă care a condus deja la lansarea coprocesoarelor. În continuare Intel va integra diferite părţi ale sistemelor de operare, pentru ca apoi să încerce integrarea unor limbaje de nivel înalt. „Aceasta va fi cu adevărat ultimul pas spre creşterea productivităţii programatorilor aplicaţii" [11].

La sfîrşitul anului 1981 Intel va mai anunţa încă trei familii de microprocesoare: *micromini*, *micromaxi* şi *micromainframe*, precum şi sistemul de operare *multitasking* RMX/86. În tabelul 1.2 sînt trecute principalele caracteristici ale celor trei noi familii împreună cu cele ale familiilor mai vechi de microcontroloare şi microcalculatoare, iar în figura 1.1 este înfăţişată evoluţia acestor produse.

Tabelul 1.2. **Familiile de microprocesoare Intel**

Clasa	Număr de biţi pe cuvînt	Preţ aproximativ	Performanţă raportată la microcontrolor		Memorie	
			UC	I/E	Mărime (în octeţi)	Mod de adresare
Micromainframe	32	$400 ÷ $3600	20 ÷ 70	6 ÷ 45	256k ÷ 8M	Adresare dinamică, segmentată sau paginată. Mecanism de memorie virtuală.
Micromaxi	16 sau 32	$100 ÷ $500	25	12	128k ÷ 1M	Adresare segmentată sau paginată. Mecanism de memorie virtuală.
Micromini	16	$20 ÷ $150	8 ÷ 10	6	32k ÷ 256k	Adresare statică, segmentată sau paginată.
Microcalculator	8 sau 16	$10 ÷ $50	1 ÷ 5	3 ÷ 5	4k ÷ 64k	Adresare segmentată sau directă.
Microcontrolor	8	$2 ÷ $20	1	1	1k ÷ 2k	Adresare directă sau absolută.

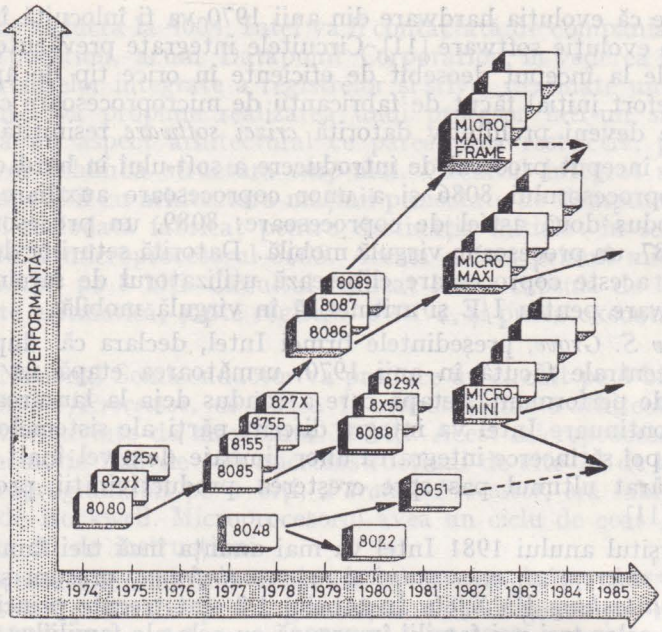


Fig. 1.1. Evoluția familiilor de microprocesoare Intel

Microcontroloarele lucrează pe 8 biți, au seturi de instrucțiuni orientate pe bit și octet și adresează maximum 64 Koct. În această clasă pot fi incluse circuitele 8022, 8048 și 8051 destinate în principiu înlocuirii logicii convenționale și/sau realizării unui control limitat. De exemplu 8051 are 32 Kbiți de ROM* în *chip* și un set de instrucțiuni capabil să trateze direct expresii booleene.

Circuitele din clasa *microcalculatoarelor* sînt destinate realizării de sisteme complete, dar de dimensiuni mici. Aceste circuite pot avea caracterstici de mașini pe 8 biți și 16 biți. De exemplu, deși 8080 și 8085 au unele instrucțiuni pe 16 biți și generează adrese de 16 biți, unitățile lor centrale lucrează pe 8 biți. 8088 are registre de 16 biți, instrucțiuni pe 8 și 16 biți, dar *bus* de I/E pe 8 biți.

Tendința de integrare a software-ului va deveni predominantă pentru clasa *micromini*. În această clasă este inclus microprocesorul pe 16 biți 8086, odată cu care a fost lansat conceptul de adresare segmentată, o facilitate ce va permite realizarea unor programe mai modulare și mai ușor de manevrat.

Clasa *micromaxi* va integra nucleul unui sistem de operare. Circuitele din această clasă, cu caracteristici de mașini pe 16 și 32 de biți, vor avea memorie protejată, permițînd accesul mai multor utilizatori.

---

* Read Only Memory, memorie fixă.

Procesorul din clasa *micromainframe* nu va mai avea nici registre la dispoziția utilizatorului, nici limbaj de asamblare. El va putea fi programat numai în limbajul de nivel înalt *ADA* şi va reprezenta un nou salt arhitectural în domeniul microprocesoarelor. *Andrew Grove* speră că acest tip de procesor va conduce la o reducere de 10 ori a numărului de programatori. În clasa *micromainframe* poate fi inclusă seria iAPX-432.

Competiția actuală între microprocesoarele pe 16 biți şi cele pe 32 de biți este, totuşi, în unele privințe, mai mult aparentă decît reală [12]. În funcție de mărimea *bus*-ului extern de date şi a registrelor interne, precum şi de posibilitățile setului de instrucțiuni, o mașină pe 16 biți poate fi la fel de flexibilă, uneori chiar mai flexibilă, decît unele mașini pe 32 de biți. În tabelul 1.3 sînt listate procesoarele pe 16 şi 32 de biți existente sau anunțate în momentul scrierii acestei lucrări. Dintre circuitele prezentate numai *MAC-32* de la Bell Telephone şi procesorul pe 32 de biți de la Hewlett-Packard sînt cu adevărat mașini pe 32 de biți.

Tabelul 1.3. **Microprocesoare pe 16 şi 32 biți**

Fabricant	Model	Tehnologie	*Bus* Date	*Bus* Adresă	Observații
DEC	LSI-11/2	NMOS	16	16	Proprietatea firmei
	LSI-11/23	NMOS	16	18	Proprietatea firmei
DG	mN601	NMOS	16	16	În stoc
	mN602	NMOS	16	16	În stoc
Fairchild	9445	I^3L	16	16	Sfîrşitul anului 1981
GI	CP1600	NMOS	16	16	În stoc
HP	HP-1000	CMOS/SOS	16	20	Proprietatea firmei
Intel	8086	NMOS(HMOS)	16	20	În stoc
	432	NMOS(HMOS)	16	24	Mostre
Motorola	MC68000	NMOS	16	24	În stoc
National	NS16016	NMOS	16	16	La începutul anului 1982
	NS16032	NMOS	16	24	Mostre
TI	TMS9900	NMOS	16	16	În stoc
	SBP9900	I^2L	16	16	În stoc
	TMS99000	NMOS	16	16	Sfîrşitul anului 1981
WD	Ada & Pascal Microengine	NMOS	16	16	În stoc
Zilog	8001	NMOS	16	23	În stoc
	8002	NMOS	16	16	În stoc
	8003	NMOS	16	23	1982
	8004	NMOS	16	16	1982
Ferranti	F100L	CDI	16	16	În stoc
Fujitsu	CMOS	CMOS	16	24	Proprietatea firmei
NEC	Pascal Pr.	NMOS	16	24	Proprietatea firmei
Philips	SP16C	NMOS	16	16	Proprietatea firmei
Toshiba	T88000	CMOS/SOS	16	24	Proprietatea firmei
Bell	MAC-32	CMOS	32	32	Proprietatea firmei
HP	32-bit	NMOS	32	32	Proprietatea firmei
IBM	320	MTL	—	—	Proprietatea firmei

MAC-32, realizat în laboratoarele Bell Telephone, este un procesor CMOS * pe 32 de biți ce disipă aproximativ 1 W. El lucrează sub sistemul de operare *Unix* și poate fi programat în *limbajul C*, ambele dezvoltate la Bell. Procesorul are *bus*-uri interne de 32 de biți și utilizează în punctele cheie redundanță triplă. Controlul procesorului este implementat cu circuite de tip PLA **.

Procesorul *single-chip* pe 32 de biți produs de firma Hewlett-Packard conține 450 000 de tranzistori. Realizat în tehnologie NMOS cu linii de 1,5 μm, el lucrează cu o frecvență a ceasului de 18 MHz, dar disipă 7 W. Avînd o arhitectură internă de tip *pipeline* microprocesorul poate executa o înmulțire de 32 × 32 biți în 1,8 μs, iar o înmulțire în virgulă mobilă pe 64 de biți în 10,4 μs.

1.3. EVOLUȚII ARHITECTURALE. PERSPECTIVE

Continua îmbunătățire a tehnologiei, creșterea gradului de integrare, reflectate în apariția microprocesoarelor, vor conduce inevitabil la inovații remarcabile în structura calculatoarelor. Posibilitatea de a integra din ce în ce mai multe funcții într-un singur *chip* îi va forța cu siguranță pe inginerii constructori de calculatoare să-și reexamineze procedurile de proiectare. Unii vor proiecta circuite integrate la comandă. Alții vor sacrifica din flexibilitate pentru a obține un preț de cost mai scăzut prin utilizarea de componente standard cum sînt microprocesoarele *bit-slice*, sau rețelele logice integrate, PLA-urile.

Datorită folosirii masive a memoriilor semiconductoare și a noilor circuite LSI va crește și gradul de utilizare a microprogramării în proiectarea calculatoarelor.

Un beneficiar important al noilor cuceriri ale tehnologiei vor fi și comunicațiile. Avînd microprocesoare, modem-urile vor deveni destul de inteligente pentru a realiza o varietate de funcții care să conducă la creșterea vitezei și siguranței comunicațiilor. Fabricanții de calculatoare vor încuraja comunicațiile de informație prin introducerea procesoarelor frontale, *front-end*, și a software-ului specializat pentru comunicații. Ca rezultat, conceptul de informatică distribuită, care are ca precursor accesul multiplu, *time-sharing*, din anii 1960, se va maturiza, permițînd dispersia funcției de procesare.

Disponibilitatea deosebită a puterii de calcul sub forma microprocesoarelor, realizarea memoriilor semiconductoare de capacitate foarte mare — tehnologii se gîndesc deja la memorii dinamice de 1 Mbit — conduc la modificări însemnate ale arhitecturii și organizării calculatoarelor, ale structurilor de control cu microprocesoare. Evoluția arhitecturală depinde în special de realizarea unor sisteme multimicroprocesor și, mai general, a unor sisteme cu mai multe elemente de procesare. De asemenea, evoluția arhitecturală este strîns legată de modificările care se fac în organizarea și administrarea memoriei, precum și a intrării/ieșirii.

* Complementary Metal Oxide Semiconductor
** Programmable Logic Array

Maşina von Neumann clasică, cea care a stat la baza calculatoarelor EDSAC şi EDVAC, nu mai poate reprezenta un model. Cercetătorii simt nevoia unor îmbunătăţiri de profunzime care să conducă la un salt din punct de vedere al vitezei, modului şi puterii de prelucrare ale maşinilor de calcul.

Prin separarea datelor de instrucţiuni s-a trecut de la organizări obişnuite de tip *SISD, Single Instruction-Single Data Stream*, la organizări de tip *SIMD, Single Instruction-Multiple Data Stream, MIMD, Multiple Instruction-Multiple Data Stream*, la arhitecturi bazate pe conceptul de *obiect, object-based architectures*, sau la arhitecturi bazate pe *fluxul datelor, data-flow architectures* [13, 14].

Organizarea *SISD* este organizarea unui calculator obişnuit, a unei maşini von Neumann clasice, care prelucrează un singur şir de date, instrucţiune după instrucţiune (fig. 1.2 *a*). Organizarea de tip *SIMD* are o singură unitate de control, care va executa la un moment dat o singură instrucţiune, dar ale cărei argumente se referă la mai multe elemente ce procesează fluxuri de date diferite (fig. 1.2 *b*). Organizarea *MIMD* are mai multe unităţi de control separate împreună cu elementele de procesare asociate. Aceste unităţi, cu accesul lor propriu la memoria calculatorului, execută instrucţiuni diferite asupra unor fluxuri de date diferite (fig. 1.2 *c*).

Un exemplu ilustrativ al evoluţiei organizatorice şi arhitecturale în domeniul microcalculatoarelor este constituit de seria iAPX-432 produsă de

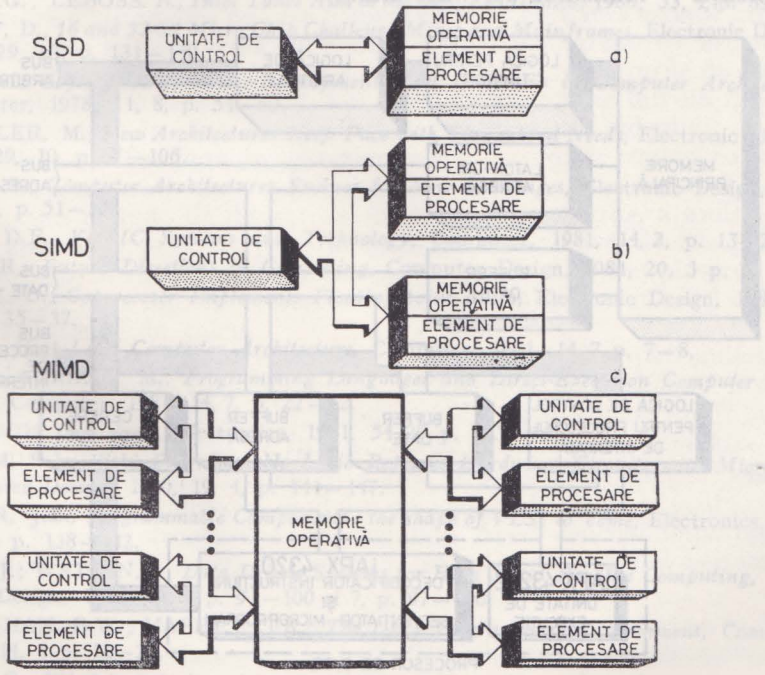


Fig. 1.2. Organizări de tip SISD, SIMD şi MIMD

Intel. Resursele hardware, setul de instrucțiuni și sistemul de operare iMAX au fost proiectate pentru a permite sistemelor realizate cu circuite din această serie utilizarea limbajului de nivel înalt ADA [14]. Circuitele din seria iAPX-432 sînt destinate implementării unor arhitecturi bazate pe conceptul de obiect. Divizarea memoriei în 16 milioane de segmente permite asignarea unui spațiu suficient fiecărui obiect important din punct de vedere al aplicației. Pentru a asigura funcționarea simultană a mai multor procesoare sau coprocesoare, circuitele au în afara *bus*-urilor de adresă și de date un *bus* de arbitraj de 8 biți (fig. 1.3). De asemenea, un *bus* de 5 biți permite conectarea procesorului de interfață într-o *fereastră* a spațiului de adresare, ceea ce oferă o posibilitate comodă de utilizare a memoriei locale de către orice subsistem. Seria iAPX-432 este compusă din circuitele iAPX 43201 și iAPX 43202, care alcătuiesc un procesor de date microprogramat, și din iAPX 43203 ce reprezintă un procesor de interfață destinat conectării la sistem a unor diverse subsisteme. Accesul la memorie și comunicația interprocesor se fac pe baza unui *descriptor de obiect* care poate fi adresat fizic de un *descriptor de acces*. În acest fel sînt controlate atît accesul la un obiect, cît și operațiile executate asupra lui.

Implicațiile integrării pe scară foarte largă sînt vaste pentru că, într-adevăr, puterea de calcul, dispozitivele de memorie au devenit extrem de ușor de obținut. Costul hardware-ului nu va mai limita realizarea, de exemplu, a unei structuri multiprocesor. Realizarea structurilor hardware ar putea deveni aproape automată. Problema care se va pune în continuare va fi adaptarea la aplicație — realizarea software-ului.

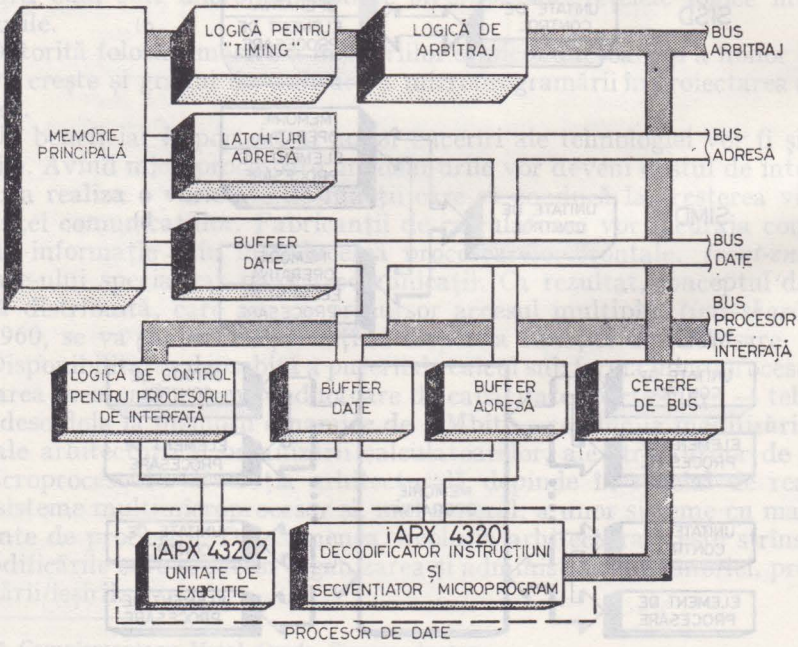


Fig. 1.3. Organizarea unei structuri de control cu circuite din familia iAPX-432

BIBLIOGRAFIE

1. TOMA, V., *Dezvoltarea construcției de mașini electronice în R.P.R. și însemnătatea lor pentru economia națională*, în *Cercetări filozofice*, 1962, IX, 6, Academia R.P.R., Institutul de filozofie.
2. TOMA, V., *L'activité dans le domaine des calculatrices électroniques digitales à l'Institut de Physique Atomique de Bucarest*. VII Rassegna internazionale electronica e nucleare, 1960.
3. TOMA, V., *The Electronic Computer of the Institute of Physics of the Academy of the R.P.R.* comunicare prezentată la Colocviul internațional de matematică, Dresda, 1955, publicată în Aktuelle Problemme der Rechentechnik, p. 27—41, Deutcher Verlag der Wissenschaften, Berlin, 1957.
4. TOMA, V., *Generarea controlată a impulsurilor în sistemele asincrone*, Teză de doctorat, Institutul Politehnic București, 1972.
5. TOMA, V., *Dispozitiv de generare controlată a impulsurilor*. Certificat de inventator nr. 45785 din 06.03.1969 cu prioritate de la 08.03.1961.
6. ȘTEFAN, I, M.; NICOLAU, E., *Scurtă istorie a creației științifice și tehnice românești*, Editura Albatros, București, 1981.
7. * * * *Colecție de programe pentru calculatorul electronic CET-500*, Editura Academiei R.S.R., București, 1967.
8. PETRESCU, M., *Dezvoltarea științei sistemelor cibernetice în România*, în *Istoria științelor în România. Cibernetica*, Editura Academiei R.S.R., București, 1981, p. 34—58.
9. DRĂGĂNESCU, M., *A doua revoluție industrială. Microelectronica, automatica, informatica —factori determinanți*, Editura tehnică, București, 1980.
10. * * * *Fifty Years of Achievment: a History*, Electronics, 1980, 53, 9, p. 36—414.
11. POSA, J.G.; LEBOSS. B., *Intel Takes Aim at the '80s*, Electronics, 1980, 53, 4, p. 89—95.
12. BURSKY, D., *16 and 32-bit Micro Chip Challenge Minis and Mainframes*. Electronic Design, 1981, 29, 10, p. 131—140.
13. LIPOVSKI, G.J.; DOTY, K.L., *Developments and Directions in Computer Architecture*, Computer, 1978, 11, 8, p. 54—60.
14. SCHINDLER, M., *New Architectures Keep Pace with Throughput Needs*, Electronic Design, 1981, 29, 10, p. 97—106.
15. BELL, G.C., *Computer Architectures Evolues for New Challenges*, Electronic Design, 1981, 29, 10, p. 51—52.
16. BARBE, D.F., *VHSIC Systems and Technology*, Computer, 1981, 14, 2, p. 13—22.
17. BELL, J.R,, *Future Directions in Computing*, Computer Design, 1981, 20, 3 p. 95—102.
18. BURSKY. D., *Coprocessor Implements Floating-point Math*, Electronic Design, 1980, 28, 10, p. 35—37.
19. CHU, Y., *High-Level Computer Architecture*, Computer, 1981, 14, 7, p. 7—8.
20. CHU, Y.; ABRAMS, M., *Programming Languages and Direct-Execution Computer Architecture*, Computer, 1981, 14, 7, p. 22—32.
21. * * * *C-MOS technology*, Electronics, 1981. 54, 20, p. 103—141.
22. DURHAM, S.J., *Field Programmable Logic Replaces Hardwired Circuits with Microcode*, Computer Design, 1980, 19. 4, p. 141—147.
23. FISCHER, J.L., *Programmable Components: the shape of VLSI to come*, Electronics, 1980, 53, 13, p. 138—142.
24. GURD, J.; WATSON, I., *Data Driven Systems for High Speed Parallel Computing*, Computer Design, 1980, 19, 6, p. 91—100 și 7, p. 97—106.
25. KERNIGHAN, B.W.; MASHEY, J.R., *The Unix Programming Environment*, Computer, 1981, 14, 4 p. 12—22.
26. POSA, J.G., *What to expect next: a special report*, Electronics, 1980, 53, 12, p. 119—129,
27. POSA, J.G., *Gate arrays: a special report*. Electronics, 1980, 53, 21, p. 145—158.

28. POSA, J.G.; ALLAN, R., *ISSCC : a gallery of gigantic memories, gigabit logic, and single-chip systems,* Electronics, 1980, 53, 4, p. 138—151.

29. RATTNER, J; LATTIN, W.W., *Ada determines architecture of 32-bit microprocessor;* Electronics, 1981, 54, 4, p. 119—126.

30. SATYANARAYANAN, M., *Multiprocessing : An Annotated Bibliography,* Computer, 1980, 13, 5, p. 101—116.

31. SCHEERDER, J., *The use of single board computers,* Electronic Engineering, 1980, 52, 643, p. 63—68.

32. SIEGEL, H.J., *A Model of SIMD Machines and Comparison of Various Interconnection Networks,* IEEE Transactions on Computers, 1979, C-28, 12, p. 907—917.

33. * * * *Technology update,* Electronics, 1980, 53, 23.

34. * * * *Technology update,* Electronics 1981, 54, 21.

35. * * * *The System 370 processor chip : a triumph for automated design,* Electronics, 1980, 53, 22, p. 139—147.

36. ZEIGLER, S; ALLEGRE, N.; JOHNSON, R.; MORRIS, J.; BURNS, G., *Ada for the Intel 432 Microcomputer,* Computer, 1981, 14, 6, p. 47—56.

PROIECTAREA STRUCTURILOR DE CONTROL CU MICROPROCESOARE

2.1. AVANTAJE ALE STRUCTURILOR DE CONTROL CU MICROPROCESOARE

După cum s-a arătat, integrarea pe scară largă — LSI și foarte largă — VLSI, SLSI * a făcut posibilă apariția unor circuite deosebit de complexe, cum sînt sistemele de microprocesoare, accesibile acum oricărui proiectant de structuri de control electronice. Existența acestor dispozitive care realizează funcții dintre cele mai complicate într-un volum foarte mic accentuează tendința ca structurile de control electronice să devină din ce în ce mai asemănătoare cu calculatoarele. Ținînd cont și de faptul că orice structură de control trebuie să prelucreze o anumită cantitate de informație pentru a asigura funcționarea corectă a sistemului pe care îl supervizează, pare încă o dată firească această echivalare a structurilor de control cu calculatoarele. De asemenea, în principiu, pentru a realiza un control cît mai eficient, structurile de control trebuie să prelucreze o cantitate cît mai mare de informație într-un mod cît mai detaliat. Evident, în zilele noastre, acest deziderat este îndeplinit cel mai bine de calculatoare. Tendința de echivalare a structurilor de control cu calculatoarele este recunoscută și în ceea ce se numește astăzi „informatică distribuită". Așa cum metoda de divizare a timpului, *time-sharing*, a anilor 1960 a dus la o creștere a posibilităților de acces la calculatoare, deci la puterea de calcul, tot așa în zilele noastre informatica distribuită, prin plasarea unui calculator mic, a unui microprocesor, la utilizator, realizează un nivel superior de acces la puterea de calcul.

Această ieșire a calculatoarelor dintr-o zonă exclusivă și pătrunderea lor ca microprocesoare spre baza piramidei informaționale, deci spre procese, spre locul de culegere a informației, impune ca din ce în ce mai mulți proiectanți de structuri de control să învețe calculatoare, microprocesoare.

O caracteristică pe care trebuie să o aibă structurile de control pentru a fi mai eficiente este flexibilitatea modului de control, lucru ce se poate realiza cel mai bine în cazul structurilor programabile. Acest concept important de *programabilitate* este strîns legat de proiectarea calculatoarelor și este ilustrat în figura 2.1 [1].

În această figură structurile de control, alcătuite în general dintr-o parte cablată, fixă, limitativă, *hardware*-ul, și o parte flexibilă, programabilă, *software*-ul

* Super Large Scale Integration.

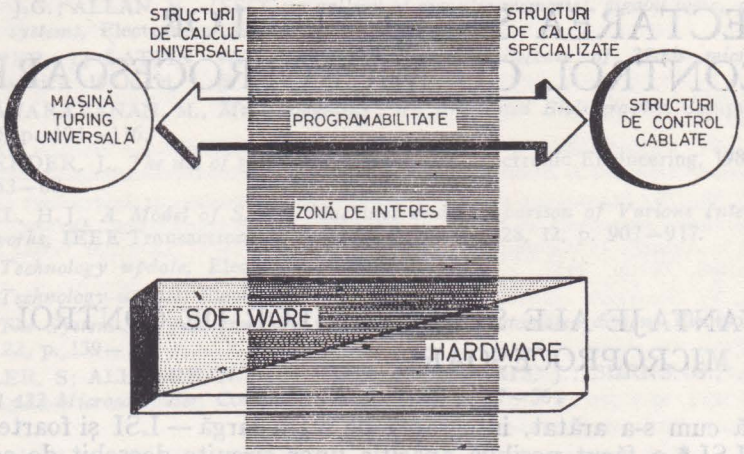


Fig. 2.1. Ilustrarea conceptului de programabilitate

sau *firmware*-ul, sînt clasificate continuu pe o singură dimensiune. La o extremă se află mașina Turing universală, cunoscută teoreticienilor de automate, un dispozitiv complet programabil, avînd un minim de hardware — probabil mașina universală cea mai generală care se poate imagina în momentul de față. La cealaltă extremă se găsește o structură de control cablată, specializată, neprogramabilă, capabilă să execute numai un singur lucru. Cheia acestei clasificări este conceptul de programabilitate. Cu cît este mai generală mașina, sau mai programabilă, cu atît setul de *instrucțiuni* necesar pentru a scrie un *program*, limbajul, este mai bogat, mai detaliat, mai precis.

Se observă că există, din punctul de vedere al flexibilității, un raport invers proporțional între hardware și software. Acest raport este deosebit de favorabil în structurile de control realizate cu microprocesoare, conferindu-le deci o foarte bună flexibilitate în raport cu mulțimea aplicațiilor.

Pentru a pune și mai bine în evidență această flexibilitate vom explica în continuare noțiunile de *organizare* și *arhitectură*, precum și legătura dintre ele în cadrul structurilor de control cu microprocesoare.

Noțiunea de organizare se referă la dispunerea fizică a componentelor, a resurselor hardware, într-o structură de control și la caracteristicile acestor componente. În sens general atributele componentelor guvernează explicit modul în care poate fi organizată structura, sistemul, și determină implicit constrîngerile funcționale ale acestora din urmă. Organizarea unei structuri tinde să reflecte mai degrabă fenomenele fizice decît pe cele logice, informatice.

Noțiunea de arhitectură se referă tocmai la aspectul logic, informatic, al unei structuri de control cu o accentuare a proprietăților interne ale acesteia, privite în special din punctul de vedere al utilizatorului. Astfel, o entitate internă, cum ar fi de exemplu un registru general sau un registru index, nu trebuie să existe efectiv. Această entitate poate fi doar un concept logic în care un cuvînt din memoria RAM a structurii de control cu microprocesor este utilizat pentru păstrarea informației corespunzătoare. Deci arhitectura

unei structuri se poate defini ca un set de resurse hardware care sînt programate sau microprogramate pentru a obține o structură informatică particulară. Arhitectura poate fi privită, de asemenea, și ca un set de constrîngeri pe care structura, sistemul, le impun *utilizatorului* [2].

Pentru o structură de control complet cablată, realizată tradițional, organizarea și arhitectura exprimă același lucru. Structurile de control cu microprocesoare, în special cele microprogramabile, permit ca o anumită organizare să suporte diverse arhitecturi.

Relația organizare-arhitectură se poate pune în evidență în mod deosebit în cazul structurilor cu microprocesoare, fiind legată de caracteristica de programabilitate pe care am discutat-o mai sus. Cu cît o structură este mai programabilă, cu atît organizarea ei suportă mai multe tipuri de arhitecturi, fiind în acest fel mai utilizabilă.

Concentrînd controlul în cîteva blocuri constructive și prezentînd un grad mare de flexibilitate, structurile de control cu microprocesoare au în comparație cu structurile hardware convenționale următoarele avantaje mari:

— costurile de fabricație ale produsului sînt mai mici;

— timpul și costul dezvoltării unor soluții originale se micșorează, deoarece proiectarea și realizarea structurilor cu microprocesoare au la bază metode mai sistematice, mai organizate;

— ca o consecință a avantajului precedent, produsele finale vor fi livrate mai repede beneficiarilor;

— flexibilitatea structurilor cu microprocesoare micșorează timpul de răspuns al producătorului la noi cerințe ale beneficiarilor, conducînd la o creștere a duratei de viață activă a produselor;

— capacitatea funcțională este mai mare la un volum și preț de cost mai mici, datorită, în primul rînd, creșterii gradului de integrare;

— fiabilitatea va fi mai bună datorită de asemenea, micșorării volumului prin integrare pe scară largă și foarte largă;

— puterea de calcul a microprocesorului poate fi utilizată și pentru autodiagnosticare, ceea ce permite reducerea timpului și costului operațiilor de service.

2.2. CONCEPTE CARE STAU LA BAZA PROIECTĂRII STRUCTURILOR DE CONTROL CU MICROPROCESOARE

După cum se știe, primii ani ai evoluției tehnologice în electronică au pus la dispoziția proiectanților de structuri numerice elemente constructive foarte simple — porți logice și bistabili —, cu putere redusă de procesare a informației.

Structurile de control numerice se proiectează tradițional prin dezvoltarea unor rețele discrete de porți și elemente de memorare care trebuie să realizeze, în final, funcțiile impuse prin temă. Rezultatul acestui mod de proiectare, descentralizat, slab coordonat, este în cele mai multe cazuri foarte complicat, greu de pus la punct, de modificat, dificil de documentat și de înțeles.

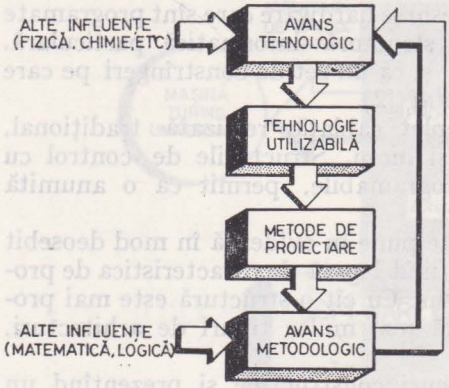


Fig. 2.2. Evoluția tehnologie-metodologie de proiectare

Proiectarea tradițională, fiind legată de un proces primar de prelucrare a informației, necesita din partea proiectantului de structuri de control cunoștințe serioase de electronică, logică booleană, teoria automatelor. În principiu proiectantul tradițional era mai mult un inginer electronist decît un proiectant de structuri sau de sisteme. Efortul său se îndrepta în general spre rezolvarea problemelor mici, trebuind să precizeze cele mai amănunțite funcțiuni hardware, corespunzătoare de fapt elementelor constructive pe care i le punea la dispoziție tehnologia.

Modul de proiectare tradițional, complicat și empiric, a evoluat urmînd o cale acumulativă de la proiectarea cu relee și tuburi cu vid, apoi cu tranzistori și circuite integrate SSI, MSI, la proiectarea modernă cu circuite LSI, VLSI. În această evoluție se poate pune în evidență legătura între tehnologie și metodologia de proiectare. Pentru a realiza un dispozitiv proiectantul încearcă să implementeze un algoritm de funcționare folosind elementele tehnologice pe care le are la dispoziție. În același timp metodele întrebuințate, care au în principiu ca scop realizarea mai sistematică, mai profesională, a unor structuri mai puternice, dar mai simple și mai economice, acționează la rîndul lor asupra tehnologiei. În acest fel se formează o entitate tehnologie-metodologie de proiectare a cărei evoluție este dată în figura 2.2 [3].

Proiectarea modernă a structurilor de control numerice, deci și a celor cu microprocesoare, se bazează pe cîteva concepte importante, cum sînt conceptul de cutie neagră, conceptele de structură și sistem, proiectare modulară, proiectare de sus în jos, electronică funcțională.

Conceptul de *cutie neagră*, *black-box*, se răspîndește odată cu apariția unor dispozitive complexe destinate unei largi utilizări ca, de exemplu, automobilul, radioul, calculatoarele [4]. Conform acestui concept obiectul nu mai interesează decît prin intrări, ieșiri și funcții realizate.

Originea etimologică a noțiunii de *structură* se găsește în limba latină, unde substantivul *structura* are înțelesul de construcție, iar verbul *struere*, care are la participiu trecut forma *structus*, înseamnă a clădi. Conceptul de structură, utilizat și de noi mai sus, este de fapt un concept bine instalat în arsenalul epistemologic al omenirii [5]. El va apărea cu o frecvență deosebită în ultimii zeci de ani, ilustrînd din punct de vedere filozofic una din căile de evoluție a cunoașterii, așa-numita *succesiune de concepte centrale*. Conceptul de structură este utilizat în zilele noastre pentru a desemna două aspecte principale. Primul aspect al conceptului se referă la *alcătuirea părților unui tot*, la starea fizică, constituția, configurația, organizarea lui. Al doilea aspect, cel evolutiv, reprezintă de fapt sensul modern al conceptului. Cuvîntul structură nu mai desemnează conform acestui sens modern o simplă configurație,

organizare, ci „un tot format din elemente solidare, în care fiecare depinde de toate celelalte şi nu poate fi ceea ce este decît în şi prin ele". „Legătura între părţi" — primul aspect — este un sens mai puţin conturat decît sistemul de *interdependenţă totală* — al fiecărei părţi cu toate celelalte —, presupus de al doilea aspect. În timp ce primul sens este o sumă, al doilea este un întreg, accentuînd faptul că suma părţilor diferă de întregul lor şi subliniind *rolul relaţiilor*, al interfeţelor, în existenţa unei structuri.

O structură de control se poate defini deci ca un set de obiecte *împreună* cu relaţiile dintre ele. Într-o structură de control cu microprocesoare obiectele sînt resursele hardware, circuitele integrate organizate în blocuri constructive. Relaţiile vor fi constituite din interfeţele logice, protocoalele, ce asigură transferul corect al informaţiei în interiorul structurii, şi din software, care împreună cu interfeţele realizează prelucrarea informaţiei, astfel încît structura să poată fi privită din exterior ca o entitate, ca o „cutie neagră".

O structură de control cu microprocesoare înseamnă atît hardware, cît şi software. De aici, legat de conceptul de structură, rezultă un alt aspect pe care dorim să îl punem în evidenţă. Este vorba despre dezvoltarea mijloacelor de programare a acestor structuri, despre elaborarea unor limbaje specifice pentru descrierea proceselor informaţionale din structurile cu microprocesoare, mai general despre elaborarea unor limbaje destinate manipulării şi prelucrării unor diverse seturi de simboluri. Aceste preocupări sînt strîns legate de matematică şi, în special, de un domeniu nou al ei, lingvistica matematică. Conceptul de structură, definit mai sus, a influenţat şi matematica permiţînd abordarea ei dintr-un alt punct de vedere, generalizat. Vom încerca să susţinem aceasta redînd cele spuse de Gr. C. Moisil în [6]. „Adevărul cel mare pentru matematicieni este că astăzi nu mai este categoria cantităţii regulatorul suprem al matematicilor. Locul i l-a luat categoria *structurii* (s.n.) ... Tehnica azi cere o matematică structurală ... În maşinile de calcul electronice digitale numărul nu apare ca rezultatul unei măsurări, ci este un număr pur. Această înţelegere a matematicii ca *ştiinţă a structurilor* (s.n.) justifică o muncă mai veche: logica matematică, şi una mai nouă: lingvistica matematică. Lingvistica structurală permite şi acceptarea unui aspect tulburător de nou al lingvisticii şi al tehnicii: intrarea lingvisticii în strînsă legătură cu tehnica."

Conceptul de *electronică funcţională*, introdus şi definit pe larg de profesorul *Mihai Drăgănescu* în [7], pune printre altele în evidenţă un mod de abordare în primul rînd funcţional şi apoi analitic-sintetic. Această abordare funcţională, arhitecturală, a electronicii este favorizată de creşterea complexităţi elementelor constructive puse la dispoziţia utilizatorilor de către tehnologie. Este posibil şi relativ comod astăzi ca proiectanţii să definească întîi funcţiile şi după aceea să le realizeze prin microelectronică. „Abordarea funcţională în electronică începe cu microprocesorul şi memoria semiconductoare." Acestea sînt elementele constructive principale cu ajutorul cărora se poate realiza o mare diversitate de funcţii informaţionale.

Cele trei concepte, de *cutie neagră*, de *structură* şi de *electronică funcţională*, considerăm că stau la baza proiectării moderne, modulare, a structurilor electronice de control, în speţă a celor cu microprocesoare.

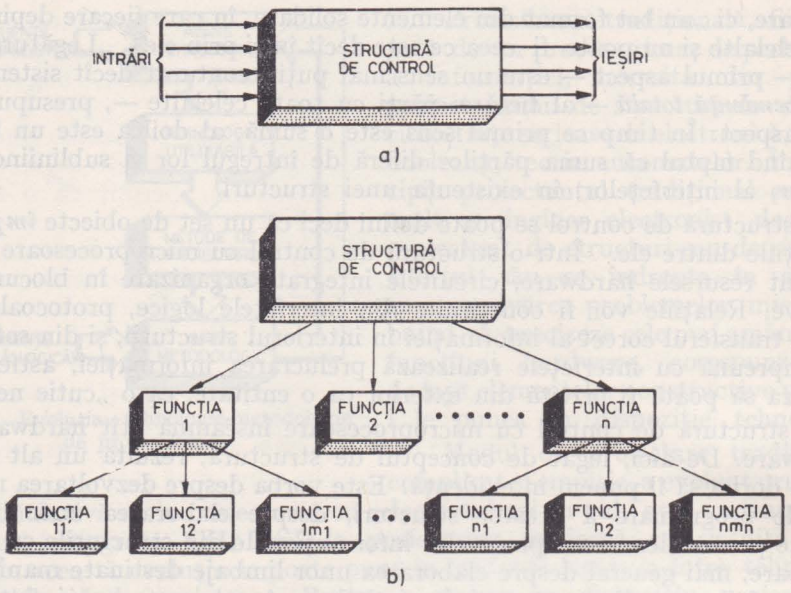


Fig. 2.3. Proiectarea modulară, de sus în jos, a unei structuri de control

La începutul procesului de proiectare structura de control, sistemul, constituie o entitate de-sine-stătătoare, o *cutie neagră*, pentru care se cunosc doar relațiile sale cu mediul înconjurător — intrările/ieșirile sau specificațiile sistemului — și funcțiile sistemului (fig. 2.3 *a*).

În continuare, conform principiului identificării funcțiilor, conceptului de electronică funcțională și abordării descendente, de sus în jos, de la simplu la complex, se trece la detalierea părților componente la structurarea sistemului pe funcții, subfuncții (fig. 2.3 *b*). Se observă abordarea arborescentă a structurii în care terminalele arborelui vor fi considerate funcțiile primitive, microoperațiile structurii.

Descompunerea structurii, sistemului, în funcții și subfuncții are în general la bază un set de criterii, dar ea depinde într-o mare măsură, ca în orice problemă inginerească, de experiența și intuiția analistului de sistem. De aceea nu se poate indica foarte precis momentul cînd se ajunge la o funcție primitivă.

Vom enumera în continuare, totuși, cîteva criterii de descompunere în funcții și subfuncții [8]:

— omogenitatea funcțiilor;
— utilizarea de resurse hardware/software standard;
— posibilitățile echipei care realizează proiectul.

În urma analizei structura de control va fi împărțită în unități, blocuri sau module, ușor de manevrat, pentru care sînt definite clar intrările, ieșirile și funcțiile. Aceste blocuri, cutii negre la rîndul lor, pot fi analizate și realizate acum independent, de proiectanți diferiți, după care vor fi integrate în structura de control finală.

Proiectarea modulară, de sus în jos, a structurilor de control cu microprocesoare presupune mai multe etape de definire, analiză, punere la punct, care în cele din urmă au scopul să sistematizeze și să simplifice activitatea de proiectare și realizare a produsului. Aceste etape, ce se referă atît la hardware cît și la software, pot fi următoarele:

— specificarea externă a sistemului;
— structurarea sistemului, adică identificarea părților componente, a funcțiilor și relațiilor între acestea;
— structurarea pe module, blocuri, corespunzătoare funcțiilor sistemului;
— specificarea modulelor în vederea proiectării lor independente, documentarea modulelor;
— măsuri administrative, distribuirea sarcinilor în cadrul echipei de proiectare;
— structurarea internă a modulelor;
— realizarea modulelor;
— punerea la punct și testarea modulelor, blocurilor;
— integrarea modulelor în cadrul sistemului;
— controlul de calitate;
— întreținerea.

În concluzie proiectarea modulară de sus în jos înseamnă dezvoltarea unei structuri elaborate, în detaliu, pornind de la o idee globală, de la o funcție principală [9]. Această procedură este o parcurgere gradată a unor etape riguros definite și bine planificate. Fiecare etapă constă în împărțirea unei probleme în subprobleme, apoi reîmpărțirea în continuare a acestor subprobleme pînă la atingerea unui nivel elementar. Procesul de proiectare, realizare și punere la punct este ciclic și iterativ.

2.3. ETAPELE PROIECTĂRII UNEI STRUCTURI DE CONTROL CU MICROPROCESOR

În acest subcapitol vom detalia pe baza celor spuse în § 2.1 și § 2.2 etapele proiectării unei structuri de control realizate cu microprocesor (fig. 2.4).

1. *Formularea problemei* este o fază esențială în procesul de proiectare. Activitățile cuprinse în cadrul acestei faze, care s-ar mai putea numi *specificarea externă a sistemului* sau *definirea sistemului*, se referă la stabilirea caracteristicilor de utilizare ale microsistemului, a funcțiilor și competențelor sale în cadrul sistemului în care va fi montat, a algoritmilor săi de bază. Insistăm asupra unei abordări atente și cît mai sistematice a acestei faze, cu riscul de a consuma un timp care poate părea nejustificat de lung în etapa incipientă a proiectului. Experiența arată însă că multe greșeli care consumă timp și necesită eventuala reformulare parțială a problemei apar la momentul cel mai nedorit tocmai din cauza unui mod de lucru pripit în faza de start a proiectului. Un aspect complementar este acela că începerea lucrului la un proiect de structură de control cu microprocesor marchează de cele mai multe ori contactul proiectantului electronist, automatist, cu un domeniu de activitate

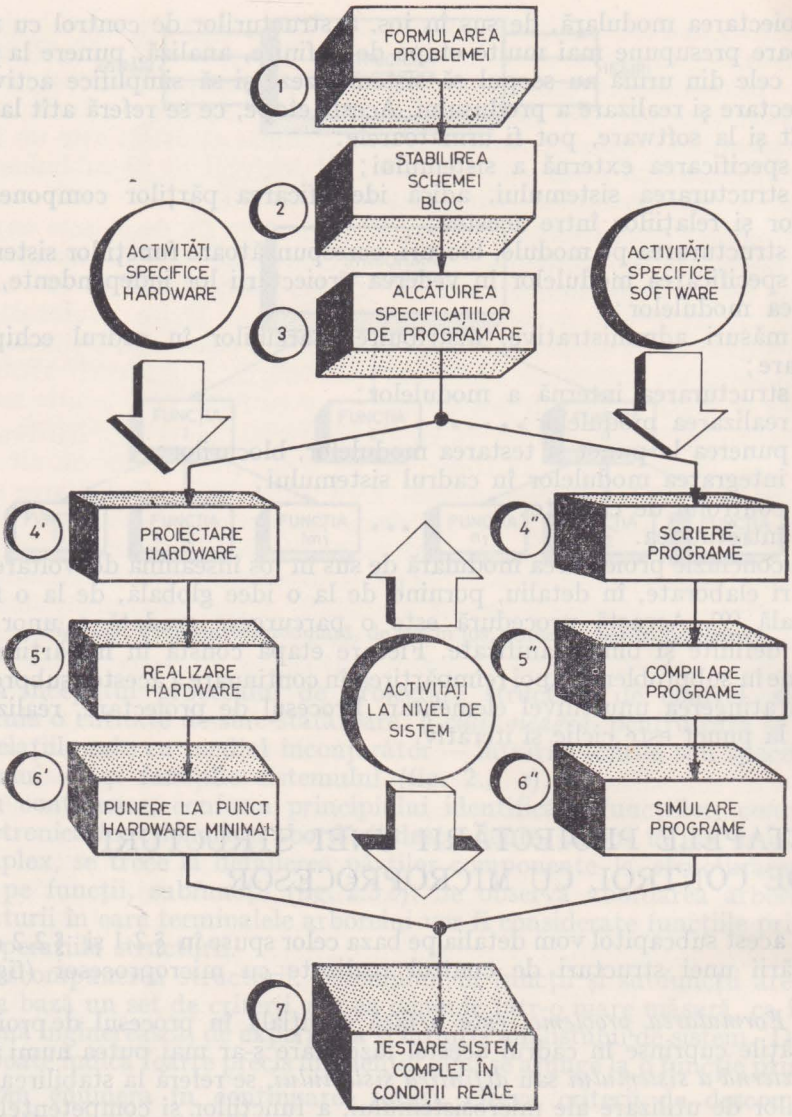


Fig. 2.4. Organigrama proiectării unei structuri de control cu microprocesor

nou pentru el. În acest sens, este foarte importantă înțelegerea corectă a problemei beneficiarului care, de regulă, nu este specialist în electronică sau automatică, familiarizarea cu terminologia domeniului, cu algoritmii de funcționare ai procesului deservit. Această interfață beneficiar-proiectant este cît se poate de importantă și asigurarea unui dialog eficient determină în mod direct reușita lucrării, gradul de utilitate al produsului final.

Rezultatul acestei faze trebuie să fie un document scris care să descrie cu precizie calităţile produsului, fără a intra în detalii de funcţionare, decît numai în măsura în care acest lucru pare indispensabil pentru claritatea textului. Documentul pe care îl numim „Specificaţiile sistemului" trebuie să întrunească atît acordul proiectantului, cît şi pe cel al beneficiarului.

2. *Stabilirea schemei-bloc.*

3. *Alcătuirea specificaţiilor de programare*

Odată ce au fost definitivate specificaţiile sistemului, ele trebuie acum transformate într-un dispozitiv logic şi un program de funcţionare care, lucrînd împreună, vor realiza funcţiile sistemului. Dacă în faza iniţială nu am fost preocupaţi de detaliile de implementare, nici de modul în care vor fi repartizate funcţiile realizate prin hardware în raport cu cele care vor fi realizate prin software, fazele 2 şi 3 se ocupă tocmai de aceste lucruri. De aceea, ele trebuie tratate aproximativ în paralel, constituind *descrierea sistemului.*

În ceea ce priveşte stabilirea schemei-bloc, care este o activitate cu specific pronunţat de hardware, recomandăm o abordare arborescentă, de sus în jos, folosind conceptul de *cutie neagră.* Din cauza enormei complexităţi a majorităţii sistemelor digitale, mintea omenească pur şi simplu nu poate imagina de la început sistemul în totalitatea detaliilor sale. Conceptul de cutie neagră, definit în §2.2, ne va permite să subdividem sistemul în părţi mai mici care să realizeze fiecare o funcţie dată. Orice astfel de subdiviziune, identificată prin funcţia realizată, va fi la acest stadiu o cutie neagră, o căsuţă în schema-bloc a sistemului, al cărei interior nu ne preocupă în această etapă a proiectării. Între căsuţele schemei-bloc se vor stabili legături care să reflecte fluxul logic al circulaţiei informaţiei, fiecare căsuţă posedînd o intrare şi o ieşire. Procesul descrierii sistemului fiind *iterativ,* compus din paşi succesivi, fiecare pas va fi constituit din partiţionarea cutiilor negre componente ale sistemului în alte cutii negre mai simple. Procesul se opreşte atunci cînd se atinge un nivel de detaliere considerat elementar de către proiectant.

Această metodologie permite *partiţionarea sistemului* într-o sumă de componente elementare, blocuri, module, care vor fi sintetizate ulterior în faza de proiectare hardware. Remarcăm de asemenea că partiţionarea se face în mod iterativ, produsul fiind descompus în final într-un arbore a cărui rădăcină este el însuşi, „frunzele" sale fiind componentele elementare. Criteriile de partiţionare nu sînt fixe, ele ţin de abilitatea şi de experienţa proiectantului; noţiunea de component elementar al sistemului este de asemenea subiectivă. Ca recomandare generală, în afara celor spuse în §2.2, este bine ca nivelul de detaliere între paşii succesivi ai partiţionării să fie astfel încît unei căsuţe să nu îi corespundă mai mult de 3 sau 4 căsuţe în schema de nivel inferior, chiar dacă proiectantul are de la început în vedere o descompunere mai largă. Respectarea acestei recomandări ţine atît de o anumită disciplină de gîndire obligatorie a proiectantului, cît şi de faptul că păstrarea la fiecare nivel a gradului de generalitate convenabil, a omogenităţii, este avantajoasă pentru identificarea celei mai bune partiţionări sau a funcţiilor comune la nivelul inferior.

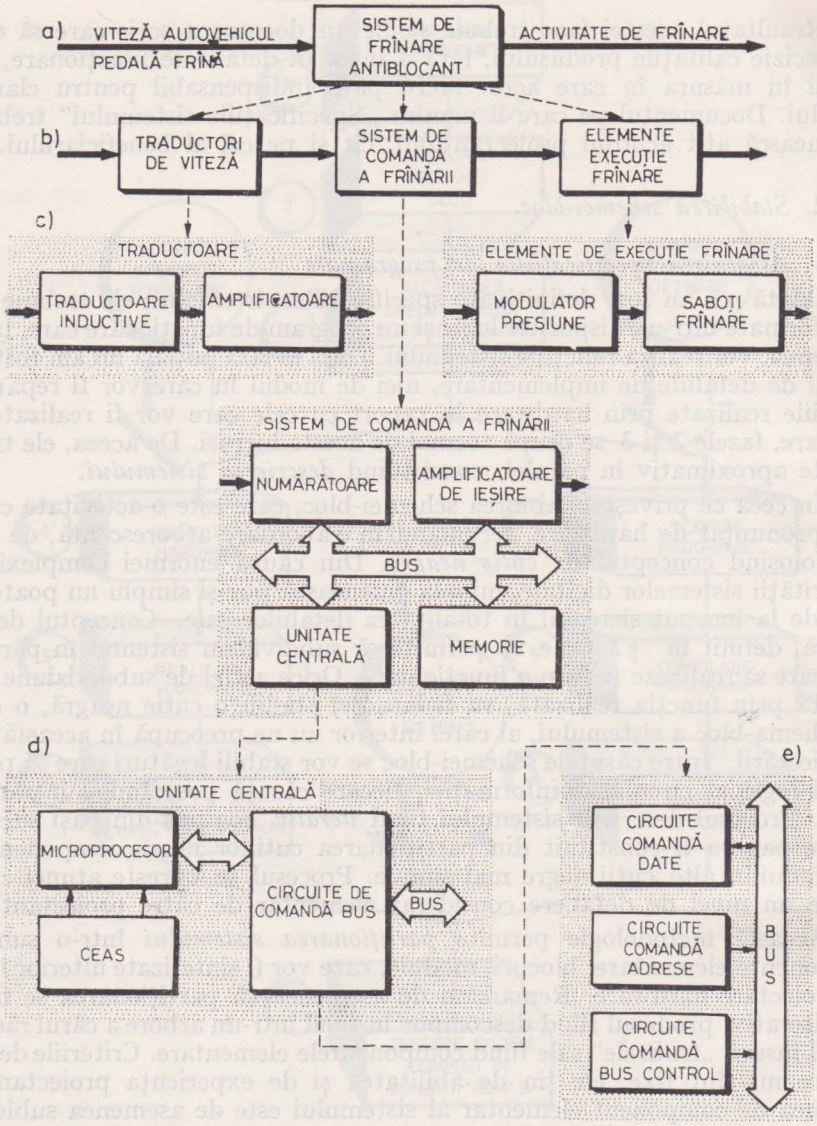


Fig. 2.5. Partiționarea arborescentă a unui sistem de frînare antiblocant controlat de un microprocesor.

Ca ilustrare a acestui mod de abordare a problemei să luăm un exemplu: partiționarea unui sistem de frînare antiblocant pentru autovehicule, realizat cu un microprocesor [10]. În figura 2.5 *a*, sistemul de frînare antiblocant este reprezentat ca o cutie neagră a cărei intrare este constituită de viteza autovehiculului și de apăsarea pe pedala de frînă, iar ieșirea este acțiunea de frînare exercitată asupra roților vehiculului. Mai departe, în figura 2.5 *b*,

ansamblul sistem de frînare antiblocant este partiționat în 3 subsisteme distincte: traductorii de viteză, sistemul de comandă a frînării și elementele de execuție-frînare. În figura 2.5 *c* și *d* procesul de partiționare continuă, oprindu-se în figura 2.5 *e* unde se consideră că s-a ajuns la un nivel de complexitate al componentelor destul de scăzut; acestea sînt componentele elementare ale schemei-bloc.

Observăm din exemplul anterior că în cursul fazei de partiționare a unei structuri cu microprocesor am obținut componente de natură hardware care, după cum se cunoaște, nu pot funcționa decît pe baza unui program ce trebuie înscris în memorie. Descrierea detaliată a programelor care implementează algoritmii de funcționare ai sistemului constituie realizarea fazei de *specificații de programare*. Rezultatul acestei faze este un document scris care precizează ce anume programe trebuie scrise, funcția exactă a fiecăruia, ce alte programe apelează în cursul execuției, ce intrări și ce ieșiri are fiecare și în ce mod le primește, respectiv transmite, și care sînt în mod estimativ necesitățile în resurse hardware: memorie PROM sau RAM, registre, timp de lucru, întreruperi etc. Pentru atingerea gradului de detaliere necesar se folosește aceeași metodă ca și în cazul schemei-bloc în care, de această dată, cutiile negre sînt module de program. Fiecare modul de program, asemănător cu o cutie neagră hardware, are un număr de intrări asupra cărora efectuează o anumită operație, pentru a obține niște ieșiri. Programul principal va avea în acest caz o structură liniară, constituind o secvență de apeluri ale modulelor de program care implementează fiecare o funcție dată. Oricare modul, conform principiului de partiționare expus anterior, se descompune în alte module mai simple, pînă ce se ajunge la un modul al cărui program pare evident. Se poate considera că modulul elementar are în jur de 50 de instrucțiuni. În programare această abordare de la simplu la complex se numește *programare modulară*, de sus în jos [8].

Un aspect important care trebuie relevat este că stabilirea schemei-bloc și realizarea specificațiilor de programare sînt două faze strîns legate una de alta și puternic interdependente. Opțiunile asupra a ceea ce trebuie implementat prin hardware și a ceea ce poate fi lăsat pe seama programului pot afecta în mod serios performanțele sistemului în forma sa finală, performanțe care la acest stadiu nu ne pot fi încă prea clare. Cu toate acestea, cîteva lucruri sînt sigure: de exemplu, implementarea unei anumite funcții prin hardware va afecta în mod pozitiv viteza de calcul a sistemului, dar îi va spori costul. Putem da aici ca exemplu tipic comparația între realizarea unei interfețe de I/E serie prin hardware și prin program. Folosind un circuit de interfață-serie specializat, un UART *, transferul unui caracter se va realiza la inițiativa microprocesorului ca urmare a executării unei singure instrucțiuni, procesorul fiind apoi degrevat de orice activitate legată de acest transfer. Prin contrast, acest lucru se poate realiza cu minimum de hardware, folosind o poartă și un bistabil, microprocesorul controlînd prin program starea liniei de intrare și comandînd în mod direct starea liniei de ieșire, care este egală cu ieșirea bistabilului; în acest caz el va fi tot timpul transferului ocupat cu realizarea

* Universal Asynchronous Receiver/Transmitter.

operației de serializare/deserializare, temporizarea biților, detecția bitului de START și alte asemenea. Alegerea celui mai bun compromis se face în funcție de cerințele de viteză, cost, fiabilitate, consum. volum ale aplicației date.

După îndeplinirea fazelor de stabilire a schemei-bloc și alcătuire a specificațiilor software este necesară o reevaluare a întregii activități depuse în cadrul acestor faze, făcînd corecțiile și completările care vor părea desigur necesare după detalierea întregului proiect. Odată definitivată descrierea sistemului se va trece pe două planuri separate și paralele la realizarea componentelor hardware și software ale sistemului.

Activitățile legate de realizarea hardware-ului sistemului sînt date mai jos.

4'. *Proiectarea hardware* are ca scop detalierea fiecărui modul de pe schema-bloc pînă la nivel de componentă electronică: circuit integrat, tranzistor, diodă, rezistență, condensator. Este deci vorba despre sinteza logică și electronică a tuturor blocurilor sistemului, documentul rezultat fiind *proiectul logic.*

5'. *Realizarea hardware.* Pe baza proiectului logic se construiesc fizic blocurile componente ale sistemului, folosind ca suport tehnologic circuitele imprimate sau plăcile universale cu socluri și conexiuni cu fire roluite, *wire-wrapping*, fundul de sertar sau cablurile panglică pentru legăturile între plăci, unul sau mai multe sertare, un sistem de conexiune între sertare, unul sau mai multe dulapuri, cabluri de legătură între dulapuri și cu dispozitivele periferice, sistemele de ventilație și de alimentare etc.

6'. *Punerea la punct a hardware-ului minimal* are ca scop asigurarea funcționării unei părți minimale a sistemului, un nucleu format din componentele indispensabile funcționării sistemului, adică: microprocesor, ceas, circuite de comandă a *bus*-urilor, un minimum de memorie. Orice operație de testare trebuie să plece de la nivelul acestui nucleu în funcțiune, pentru că în absența oricăreia dintre componentele amintite nu se poate asigura executarea nici unei instrucțiuni de program. Tot în cadrul acestei faze pot fi puse la punct și acele părți ale sistemului care nu au nevoie de intervenția programului pentru a funcționa.

Pe partea de activități software se derulează în paralel etapele descrise în continuare.

4''. *Scrierea programelor* care implementează algoritmii de funcționare a sistemului; activitatea are ca punct de plecare specificațiile de programare alcătuite în cursul fazei 3.

La acest nivel fiecare modul va fi transformat în instrucțiuni folosind un anumit limbaj de programare: limbaj de nivel înalt cum ar fi: BASIC, PL/M, PASCAL, FORTRAN sau limbaj de asamblare. De o deosebită importanță pentru înțelegerea și punerea la punct ulterioară a programelor este ca ele să fie cît mai bine comentate, folosind posibilitățile fiecărui limbaj de programare de a insera comentarii personale printre instrucțiuni. În această direcție

trebuie pornit de la ideea ca *listing*-ul programului să fie ulterior unicul document necesar.

5''. *Compilarea programelor* este faza în care se transformă programele obținute la 4'' în coduri-obiect executabile pe sistemul dedicat. În același timp se obține și listing-ul care va constitui în continuare documentul de lucru pentru activitățile software.

6''. *Simularea programelor.* În această etapă are loc o primă testare parțială a funcțiilor software. Pentru că sistemul final nu este încă disponibil, acest lucru trebuie făcut prin simulare; există două moduri de simulare posibile: simularea software pe un calculator mare, care *emulează*, prin programele sale, structura și instrucțiunile microprocesorului folosit, și simularea pe un sistem de dezvoltare realizat cu același tip de microprocesor. Verificarea modulelor de program va avea loc în pași succesivi, modulele a căror funcționare nu este încă testată fiind înlocuite cu rutine „de probă" care sînt confecționate *ad-hoc* din cîteva instrucțiuni. De exemplu, într-un sistem de reglare a temperaturii se primesc intrări numerice de la niște termocuple; înainte de a lua orice decizie asupra evoluției sistemului termic aceste date numerice de intrare trebuie normalizate. În cursul testării programului principal nu ne vom preocupa prea mult de rezultatul numeric furnizat de rutina de normalizare pe care o vom face să livreze un șir de constante extrase dintr-o tabelă construită special pentru test. La rîndul ei rutina de normalizare va fi testată ulterior independent de restul progamului.

7. *Testarea sistemului în condiții reale.* Este faza finală în care se verifică performanțele sistemului și satisfacerea integrală a specificațiilor care au fost stabilite în faza inițială. Structura cu microprocesor, complet realizată din punct de vedere fizic, cu toate programele de funcționare încărcate în memorie, este încercată în condiții reale, căutîndu-se epuizarea tuturor situațiilor practic posibile. Se fac acum ultimele ajustări ale proiectului logic și ale programelor, la sfîrsitul acestei faze microsistemul putînd fi predat beneficiarului.

BIBLIOGRAFIE

1. SALISBURRY, A.B., *Microprogrammable Computer Architecture*, Elsevier, New York, 1976.
2. LEIS, C.T., *Microcomputer Hardware Architecture*, Proceeding of IEEE, 1973, 61, 11.
3. LUPU, C.; STĂNCESCU, ȘT.; DIMITRIU, B.; IONESCU, C., *Metodologii de proiectare a structurilor de control numerice*, 4th International Conference on Control Systems and Computer Science, Bucharest, 1981, II, p. 60—65.
4. KATZAN, JR.,H., *Microprogramming Primer*, Mc Graw-Hill Book Co., New York, 1977.
5. NEMOIANU, V., *Structuralismul*, Editura pentru Literatură Universală, București, 1967, p. 18—28.

6. MOISIL, GR.C., *Structuralismul și dialectica*, în *Secolul 20*, 1967, 5, p. 97—102.
7. DRĂGĂNESCU, M., *A doua revoluție industrială. Microelectronica, automatica, informatica —factori determinanți*, Editura tehnică, București, 1980.
8. VĂDUVA, I.; BALTAC, V.; FLORESCU, V.; FLORICICĂ, I., *Programare structurată*, Editura tehnică, București, 1978.
9. CAPLENER, H.D.; JANKU, J.A., *Top-down Approach to LSI System Design*, Computer Design, 1974, 13, 8, p. 143—148.
10. CARP, V.; LUPU C., *Contribuții la realizarea blocului electronic de comandă al dispozitivului antiblocant pentru autovehicule rutiere*, lucrările Sesiunii de comunicări ICPTT, București, 1981, p. 355—362.
11. GLADSTONE, B.E., *Comparing Microcomputer Development System Capabilities*, Computer Design, 1979, 18, 2, p. 83—90.
12. McCANN, T., *Automatic software assembly*, Electronic Engineering, 1980, 52, 639, p. 51—57.
13. SAPONAS, T., *Development system: soft keys+four buses=easy use and full-speed emulation*, Electronic Design, 1979, 27, 20, p. 38—42.
14. * * * Special Report: *Microcomputer Development Systems*, Electronic Design, 1979, 27, 19, p. 92—107.

MICROPROCESOARE *SINGLE-CHIP*

3.1. PREZENTARE GENERALĂ

Realizarea de vîrf a ultimului deceniu în tehnologia circuitelor integrate o constituie microprocesorul. Diversitatea și complexitatea produselor existente în momentul de față nu ne permit să dăm o definiție detaliată și în același timp suficient de generală microprocesorului.

Structura cea mai simplă de microprocesor se obține prin integrarea funcțiilor unui procesor tradițional într-un singur circuit LSI sau VLSI. Pentru a realiza un calculator, denumit microcalculator, cu ajutorul unui microprocesor, mai sînt necesare elemente de memorie, circuite de I/E etc.

Microprocesoarele au cunoscut o evoluție foarte rapidă, din momentul apariției primului microprocesor — iunie 1971, Intel 4004 — pînă în prezent.

Atît necesitățile practice rezultate din gama diversă de aplicații, cît și concurența dintre firmele producătoare, au imprimat un ritm rapid de perfecționare a microprocesoarelor și circuitelor secundare necesare realizării microcalculatoarelor.

Direcțiile principale urmărite în perfecționarea microprocesoarelor sînt:

— Creșterea gradului de integrare, a numărului de tranzistoare înglobate într-o pastilă de siliciu, determinînd scăderea numărului de componente necesare realizării unui microcalculator; acest lucru mărește fiabilitatea, scade prețul de cost și micșorează efortul de proiectare a unui microcalculator.

— Reducerea consumului de putere.

— Mărirea vitezei de lucru.

— Compatibilitatea software dintre noile microprocesoare și cele vechi; acest fapt este impus de necesitatea utilizării eforturilor software anterioare realizării microprocesorului în cauză.

În continuare vom analiza evoluția microprocesoarelor realizate în decursul anilor de firma Intel, una din cele mai cunoscute firme producătoare de circuite LSI, VLSI, pe plan mondial. În tabelul 3.1 este dată viteza de lucru în μs pentru execuția principalelor operații de către microprocesoarele 8008, 8080 și 8086, în tabelul 3.2 sînt arătate principalele caracteristici tehnologice ale microprocesoarelor 8008, 8080, 8085 și 8086 [16], iar în tabelul 3.3 sînt prezentate comparativ caractersticile de structură generală ale acestora [16].

Tabelul 3.1. **Viteza de lucru a 3 microprocesoare Intel în μs**

Operația	Microprocesor		
	8008 (1972)	8080 (2 MHz) (1974)	8086 (8 MHz) (1978)
Transferul informației între registre	12,5	2	0,25
Operații de salt	25	5	0,875
Operații imediate cu registre	20	3,5	0,5
Apelul unei subrutine	28	9	2,5
Incrementarea unei date de 16 biți	50	2,5	0,25
Adunarea pe 16 biți	75	5	0,375
Transferul unei date de 16 biți	25	2	0,25

Tabelul 3.2. **Caracteristici tehnolcgice a 4 microprocesoare Intel.**

Caracteristica	Microprocesor			
	8008 (1972)	8080 (1974)	8085 (1976)	8086 (1978)
Tehnologia	PMOS	NMOS	NMOS	NMOS (HMOS)
Frecvența de lucru	0,5—0,8 MHz	2—3 MHz	3—5 MHz	5—8 MHz
Întîrzierea minimă pe poartă	30 ns	15 ns	4 ns	3 ns
Produsul viteză-putere	100 pJ	40 pJ	10 pJ	2 pJ
Numărul aproximativ de tranzistoare	2 000	4 500	6 500	29 000
Suprafața medie a tranzistoarelor în miimi de *inch* pătrat pe tranzistor	8,4	7,5	5,7	2,5

Tabelul 3.3. **Caracteristici de structură generală a 4 microprocesoare Intel**

Caracteristica	Microprocesor			
	8008	8080	8085	8086
Număr instrucțiuni	48	78	80	97
Număr indicatori de condiții	4	5	5	9
Capacitatea maximă a memoriei	16 Kocteți	64 Kocteți	64 Kocteți	1 Moctet
Număr maxim de ø*port*-uri de I/E	4 ieșiri 18 intrări	256 ieșiri 256 intrări	256 ieșiri 256 intrări	64 Kieșiri 64 Kintrări
Număr pini	86	40	40	40
Mărimea *bus*-ului de adrese	*	16	16*	20*
Mărimea *bus*-ului de date	8*	8	8*	16*
Tipuri de date cu care poate lucra	8 biți fără semn	8 biți fără semn, 16 biți fără semn, BCD împachetat	8 biți fără semn, 16 biți fără semn, BCD împachetat	8 biți fără semn, 8 biți cu semn, 16 biți fără semn 16 biți cu semn, BCD împachetat, BCD neîmpacheatt

* *Bus*-urile de date și adrese sînt multiplexate.

În paralel cu proiectarea și lansarea în fabricație a unor noi tipuri de microprocesoare, firmele producătoare au perfecționat atît microprocesoarele existente, cît și circuitele necesare realizării unui microcalculator în ideea măririi vitezei de lucru a acestuia. În tabelul 3.4 sînt prezentate în acest sens, cîteva exemple de microprocesoare pe 16 biți [3].

Tabelul 3.4. **Evoluția unor microprocesoare**

Firma producătoare	Tip microprocesor	Numărul de pini pe *chip*	Frecvența ceasului în MHz	Capacitatea memoriei direct adresabile	Timpul de adunare a conținutului a 2 registre	Timpul de adunare a conținutului unui registru cu o locație de memorie
Intel	iAPX86/88	40	5	1 Moctet	600 ns	2,6 μs
			8		375 ns	1,625 μs
			10		300 ns	1,3 μs
Motorola	MC 68000	64	4	16 Mocteți	1 μs	2,25 μs
			6		750 μs	1,69 μs
			8		500 ns	1,125 μs
			10		400 ns	900 ns
Zilog	Z8001	48	4	8 Mocteți	1 μs	
			6		667 ns	
			10		400 ns	
	Z8002	40	4	64 Kocteți	1 μs	
			6		667 ns	
			10		400 ns	

Prima tehnologie folosită pentru fabricarea microprocesoarelor a fost tehnologia PMOS care permite o densitate mare, în condițiile unei viteze de lucru relativ scăzute. Una din tehnologiile larg utilizate în prezent este tehnologia NMOS ce asigură o viteză de lucru sporită în comparație cu PMOS.

O tehnologie ce cîștigă teren în realizarea unor procesoare cît mai performante este și tehnologia CMOS. Aceasta utilizează în implementarea unei celule elementare o combinație formată dintr-un tranzistor NMOS și unul PMOS. Caracteristicile de viteză și densitate obișnuite sînt situate între cele ale tehnologiilor PMOS și NMOS. Printre calitățile principale ale tehnologiei CMOS se numără consumul redus de putere și imunitatea mai mare la zgomote. Tehnologia HMOS, în fapt o tehnologie NMOS îmbunătățită, permite realizarea unor circuite cu un grad înalt de integrare și consum redus de putere în raport cu circuitele în tehnologie NMOS tradițională, cu funcții similare.

În momentul de față proiectanții și firmele producătoare caută să reproiecteze microprocesoarele și circuitele necesare realizării unui microcalculator din tehnologia NMOS în tehnologia HMOS. Astfel, firma Intel a reproiectat microprocesorul 8085A/8085A-2 și circuitele necesare acestuia în tehnologia HMOS. Un microprocesor 8085AH/8085AH-2 în tehnologie HMOS are aceleași performanțe și caracteristici cu ale microprocesorului 8085A/8085A-2, dar un consum de putere mai mic cu 30% decît acesta [10].

Tabelul 3.5. **Microprocesoare** *single-chip*

Firma de origine	Microprocesorul	Tehnologie	Mărime cuvînt (date/instrucțiune)	Capacitate memorie adresată direct
Fairchild	F8	NMOS	8/8	64K
General Instrument	8000	PMOS	8/8	1K
Intel	8008	PMOS	8/8	16K
Intel	8035/8039	NMOS	8/8	64K
Intel	8080A	NMOS	8/8	64K
Intel	8085	NMOS	8/8	64K
MOS Technology	MCS-650X	NMOS	8/8	64K
MOS Technology	MCS-651X	NMOS	8/8	64K
MOSTEK	3874	NMOS	8/8	64K
Motorola	6800	NMOS	8/8	64K
Motorola	6802/6808	NMOS	8/8	64K
Motorola	6803	NMOS	8/8	64K
Motorola	6809	NMOS	8/8	64K
National Semiconductor	INS8060	NMOS	8/8	4K
National Semiconductor	NSC800	CMOS	8/8	64K
NEC Microcomputers	μPD 8080A	NMOS	8/8	64K
RCA	1802	CMOS	8/8	64K
RCA	8085AC	CMOS	8/8	64K
Signetics	2650	NMOS	8/8	32K
Zilog	Z80	NMOS	8/8	64K
Intersil	6100	CMOS	12/12	4K
Toshiba	T3190	PMOS NMOS	12/12	4K
AMD	Am 29116	ECL	16/16	64K
Data General	mN601	NMOS	16/16	32K
Data General	mN602	NMOS	16/16	64K
Fairchild	9440	I^3L	16/16	64K
Fairchild	9445	I^3L	16/16	64K
General Instrument	CP1600/1610	NMOS	16/16	64K
Ferranti	F100L	Bipolar	16/16	32K
Intel	8086	NMOS	16/16	1M
Intel	8088	NMOS	16/16[1]	64K
Motorola	MC68000	NMOS	16/16	16M
National Semiconductor	INS8900	NMOS	16/16	64K
National Semiconductor	NS16008	NMOS	16/16[1]	64K
National Semiconductor	NS16016	NMOS	16/16	64K
National Semiconductor	NS16032	NMOS	16/16	16M
Panafacom	NM1610	NMOS	16/16	64K
Texas Instruments	TMS9980/9981	NMOS	16/16[1]	8K
Texas Instruments	TMS9985	NMOS	16/16[1]	32K
Texas Instruments	TMS/SBP9900	NMOS/ I^2L	16/16	32K
Western Digital	WD-16	NMOS	16/16	64K
Zilog	Z8000	NMOS	16/16	64K 8M

NOTE.

1. *Bus*-ul extern are 8 biți, iar *bus*-ul intern 16 biți.
2. Numai 9980.
3. Cu excepția liniilor de ceas.

Număr instrucțiuni	Compatibilitate TTL	Aritmetică BCD	Controlor întreruperi inclus/niveluri întreruperi	Număr de registre generale	Număr de registre pentru stivă	Generator de ceas inclus	Compatibilitate DMA	Circuite specializate de memorie și I/E	Număr conexiuni (pini)	Tensiuni de alimentare (V)
69	Da	Da	Da/1	64	RAM	Da	Da	Da	40	5, 12
48	Nu	Da	Da/1	48	0	Nu	Nu	Da	40	5, −12
48	Nu	Da	Da/1	6	7 × 14	Nu	Nu	Da	18	5, −9
96	Da	Da	Da/1	64	RAM	Da	Da	Da	40	5
78	Da[3]	Da	Da/1	8	RAM	Nu	Da	Da	40	5, 12, −5
80	Da	Da	Da/4	8	RAM	Da	Da	Da	40	5
56	Da	Da	Da/1	0	RAM	Da	Nu	Da	40	5
56	Da	Da	Da/1	0	RAM	Nu	Nu	Da	40	5
70	Da	Da	Da/4	64	RAM	Da	Nu	Da	40	5
72	Da	Da	Da/1	0	RAM	Nu	Da	Da	40	5
72	Da	Da	Da/1	128/0	RAM	Da	Da	Da	40	5
82	Da	Da	Da/1	128	RAM	Da	Da	Da	40	5
59	Da	Da	Da/1	0	RAM	Da	Da	Da	40	5
46	Da	Da	Da/1	8	RAM	Da	Da	Da	40	5
150	Da	Da	Da/15	14	RAM	Da	Da	Da	40	5
78	Da[3]	Da	Da/1	8	RAM	Nu	Da	Da	40	5, 12, −5
91	Da	Da	Da/1	16	RAM	Da	Da	Da	40	3 ÷ 12
80	Da	Da	Da/4	8	RAM	Da	Da	Da	40	5
75	Da	Da	Da/1	7	8 × 15	Nu	Da	Da	40	5
158	Da	Da	Da/1	14	RAM	Nu	Da	Da	40	5
81	Da	Nu	Da/1	0	RAM	Da	Da	Da	40	4 ÷ 11
108	Da	Nu	Da/8	8	RAM	Da	Da	Da	36	5, −5
30	Da	Nu	Nu	32	32	Nu	Da	Da	48	5
42	Da	Nu	Da/1	4	RAM	Da	Da	Da	40	5, 10, 14, −4,25
82	Da	Nu	Da/16	4	RAM	Nu	Da	Da	40	3, 12, ±5
42	Da	Da	Da/16	4	RAM	Da	Da	Da	40	5
100	Da	Nu	Da/16	4	RAM	Nu	Da	Da	40	5
87	Da	Nu	Da/1	8	RAM	Nu	Da	Da	40	5, 12. −3
153	Da	Nu	Da/1	RAM	RAM	Nu	Da	Da	40	5, 12
97	Da	Da	Da/1	8	RAM	Da	Da	Da	40	5
97	Da	Da	Da/1	8	RAM	Da	Da	Da	40	5
61	Da	Da	Da/1	16	RAM	Nu	Da	Da	64	5
45	Da	Da	Da/6	4	10 × 16	Nu	Da	Da	40	5
78/110	Da	Da	Da	8	RAM	Nu	Da	Da	40	5
78/100	Da	Da	Da	8	RAM	Nu	Da	Da	40	5
100	Da	Da	Da	8	RAM	Nu	Da	Da	48	5
33	Da[3]	Nu	Da/3	5	RAM	Nu	Da	Da	48	5
69	Da[3]	Nu	Da/4	16	RAM	Da	Da	Da	40	5, 12, −5[2]
68	Da	Nu	Da/4	RAM	RAM	Da	Da	Da	40	5
69	Da[3]	Nu	Da/16	16	RAM	Nu	Da	Da	64	5, 12, −5
116	Da	Da	Da/16	6	RAM	Nu	Da	Da	40	5, 12, −5
110+	Da	Da	Da/1	16	RAM	Nu	Da	Da	40/48	5

Tabelul 3.6. **Microcalculatoare** *single-chip*

Firma de origine	Microcalculator	Tehnologie	Mărime cuvînt (date/instrucțiune)	Mărime memorie RAM	Mărime memorie ROM/PROM	Posibilități de extensie a memoriei
Fairchild	F38E70	NMOS	8/8	64×8	2048×8	Da
	F3878	NMOS	8/8	64×8	4096×8	Da
General Instrument	PIC1645	NMOS	8/12	24×8	256×8	Nu
	1650	NMOS	8/12	32×8	512×8	Nu
	1655	NMOS	8/12	32×8	512×8	Nu
	1670	NMOS	8/12	48×8	1024×8	Nu
Intel	8021	NMOS	8/8	64×8	1024×8	Nu
	8022	NMOS	8/8	64×8	2048×8	Nu
	8041/8741	NMOS	8/8	64×8	1024×8	Da
	8048/8748	NMOS	8/8	64×8	1024×8	Da
	8049	NMOS	8/8	128×8	2048×8	Da
Intersil	87C41	CMOS	8/8	64×8	1024×8	Da
	80C48/87C48	CMOS	8/8	64×8	1024×8	Da
Mostek	3870	NMOS	8/8	64×8	2048×8	Da
	3872	NMOS	8/8	128×8	4096×8	Da
	3873	NMOS	8/8	64×8	2048×8	Da
	3876	NMOS	8/8	128×8	2048×8	Da
Motorola	6801/6870 1	NMOS	8/8	128×8	2048×8	Da
	6805/705	NMOS	8/8	64×8	1100×8	Da
	6805R2	NMOS	8/8	64×8	2048×8	Da
	146805	CMOS	8/8	64×8	1100×8	Da
National Semiconductor	INS8050	NMOS	8/8	256×8	4096×8	Da
	INS8072	NMOS	8/8	64×8	2560×8	Da
RCA	COP1804	CMOS/SOS	8/8	64×8	2048×8	Da
Rockwell	PPS-8	PMOS	8/8	0	0	Da
	PPS-8/2	PMOS	8/8	0	0	Da
	R6500/1	NMOS	8/8	64×8	2048×8	Da
Zilog	Z8	NMOS	8/8	144×8	2048×8	Da
Texas Instruments	TMS 9940E/9940M	NMOS	①	128×8	2048×8	Nu
Intel	2920	NMOS	25/25	40×25	192×24	Nu

NOTĂ. [1] Extern 8 biți și intern 16 biți.

În ultimii ani asistăm la o accelerare rapidă a utilizării microprocesoarelor în cît mai multe și mai diverse aplicații. Acest lucru a impus proiectanților perfecționarea și diversificarea microprocesoarelor, în așa fel ca acestea să satisfacă o mare gamă de cerințe ale utilizatorilor. Dispozitivele realizate cu ajutorul microprocesoarelor au progresat de la simple blocuri de control la dispozitive de comandă foarte complicate și chiar adevărate calculatoare.

În acest sens microprocesoarele se pot împărți în 3 categorii principale. În prima categorie intră *microprocesoarele pe 8 biți* care înglobează într-un singur *chip* funcțiile unității centrale dintr-un sistem de calcul obișnuit, ele fiind microprocesoarele cu cea mai largă răspîndire și utilizare pînă în momentul de față. Microprocesoarele tipice din această categorie sînt: 8080, 8085A, Z80 și MC 6800. În tabelul 3.5 s-au prezentat cîteva microproce-

Număr instrucțiuni	Frecvență maximă de ceas (KHz)	Generator de ceas inclus	Compatibilitate TTL	Aritmetică BCD	Controlor întreruperi inclus/niveluri întreruperi	Număr de registre generale	Număr linii de I/E	Număr de conexiuni (pini)	Tensiuni de alimentare (V)
70	4 000	Da	Da	Da	Da/4	RAM	32	40	5
70	4 000	Da	Da	Da	Da/4	RAM	32	40	5
30	1 000	Da	Da	Da	Da/1	RAM	4	24	5
30	1 000	Da	Da	Da	Da/1	RAM	32	40	5
30	1 000	Da	Da	Da	Da/1	RAM	8	28	5
30	1 000	Da	Da	Da	Da/1	RAM	32	40	5
70	3 000	Da	Da	Da	Da/1	RAM	21	28	5
70	3 000	Da	Da	Da	Da/1	RAM	27	40	5
90	6 000	Da	Da	Da	Da/1	RAM	18	40	5
96	6 000	Da	Da	Da	Da/1	RAM	27	40	5
96	11 000	Da	Da	Da	Da/1	RAM	27	40	5
90	6 000 (5V)	Da	Da	Da	Da/1	16+RAM	18	40	5÷10
96	6 000 (5V)	Da	Da	Da	Da/1	RAM	27	40	5÷10
70	4 000	Da	Da	Da	Da/4	RAM	32	40	5
70	4 000	Da	Da	Da	Da/4	RAM	32	40	5
70	4 000	Da	Da	Da	Da/4	RAM	32	40	5
70	4 000	Da	Da	Da	Da/4	RAM	32	40	5
82	3 580	Da	Da	Da	Da/1	RAM	31	40	5
61	3 580	Da	Da	Da	Da/1	RAM	20	28	5
61	3 580	Da	Da	Da	Da/1	RAM	20	28	5
61	3 580	Da	Da	Da	Da/1	RAM	20	28	5
96	11 000	Da	Da	Da	Da/1	RAM	27	40	5
74	4K	Da	Da	Da	Da/2	RAM	0	40	5
113	8 000	Da	Da	Da	Da/1	RAM	13	40	5÷10
100	256/4	Nu	Nu	Da	Da/3	2	15	42	−17/+5, −12
100	200/4	Nu	Nu	Da	Da/3	2	15	42	−17/+5, −12
56	2 000	Da	Da	Da	Da/1	RAM	32	40	5
47	8 000	Da	Da	Da	Da/6	RAM	32	40	5
68	5 000	Da	Da	Da	Da/4	RAM	16	40	5
21	2 500	Da	Da	—	—	RAM	12	28	5, −5

soare din această categorie, împreună cu principalele caracteristici ale acestora [6].

În cea de-a doua categorie sînt incluse *microprocesoarele pe 16 biți* care sînt asemănătoare din punctul de vedere al funcțiilor cu microprocesoarele pe 8 biți, avînd însă lungimea cuvîntului, date/instrucțiune, de 16 biți. Dintre microprocesoarele pe 16 biți, cele mai cunoscute și utilizate sînt: 8086, Z8000 și MC68000. În a treia categorie intră microprocesoarele pe 4/8 biți, sau *microcalculatoarele single-chip*, ce conțin într-un singur circuit integrat, alături de procesor, și *port*-uri de I/E, precum și memorie de tip RAM* și PROM**.

* Random Access Memory, memorie cu conținut aleator.

** Programmable ROM, memorie fixă programabilă.

Principalul domeniu de aplicaţie al acestor microprocesoare îl constituie automatele programabile. Pentru cerinţe modeste aceste dispozitive pot fi de-sine-stătătoare, microsistemul constă atunci dintr-un singur circuit — microprocesorul. Pentru aplicaţii mai complexe, care cer flexibilitate şi facilităţi suplimentare, la acest tip de circuit se conectează *port*-uri suplimentare de I/E şi memorie externă.

În tabelul 3.6 sînt prezentate cîteva asemenea microprocesoare cu principalele lor caracteristici [6]. Deoarece microprocesoarele pe 4 biţi sînt folosite pe o scară foarte mică ele nu vor mai fi prezentate în această lucrare.

În viitorul apropiat va apărea o nouă categorie de microprocesoare — *microprocesoarele pe 32 de biţi*. Pînă în prezent au fost anunţate două microprocesoare pe 32 biţi de către Bell Telephone —MAC32— şi Hewlett Packard. Se preconizează ca în următorii ani acest tip de microprocesoare să acopere o mare parte din aplicaţiile ce se rezolvă în momentul de faţă cu ajutorul minicalculatoarelor şi chiar să le înlocuiască. Pentru anii 1982—1985 firmele Intel şi Zilog au anunţat şi ele apariţia unor procesoare pe 32 biţi.

În momentul de faţă microprocesorul *NS 16000* produs de firma National are *bus*-ul extern de 16 biţi, dar lucrează intern cu cuvinte a căror lungime este de 32 biţi. El este considerat un intermediar între microprocesoarele pe 16 biţi şi cele pe 32 biţi. Seria Z8000 lucrează cu date a căror lungime este cuprinsă între 8 şi 64 biţi, ca rezultat al flexibilităţii unităţii de prelucrare.

Cu toată dezvoltarea rapidă a microprocesoarelor pe 16 biţi în prezent continuă să apară variante noi sau îmbunătăţite de microprocesoare pe 8 biţi. Cele mai noi microprocesoare pe 8 biţi sînt structurate astfel încît ele asigură execuţia directă a programelor scrise într-un limbaj de nivel înalt, cum este PASCAL.

S-au produs microprocesoare performante ca MC6809, de către firma Motorola, Intel 8088 sau microprocesoare de mică putere bazate pe tehnologia CMOS.

Microprocesoarele tipice în tehnologie CMOS sînt: National NSC800, Motorola MC146805 şi RCA1802.

Microprocesorul NSC800, produs de firma National Semiconductor, poate executa setul de instrucţiuni al microprocesorului Z80, avînd însă o arhitectură şi o configuraţie a pinilor diferite; el poate funcţiona la frecvenţa maximă de 4 MHz.

Configuraţia minimală, formată din 3 *chip*-uri, microprocesorul, un circuit de memorie şi unul de I/E, disipă numai 125 mW.

Alt microprocesor de 8 biţi este *8088* care are performanţe de două ori mai bune decît microprocesorul 8085. 8088 menţine arhitectura internă pe 16 biţi a microprocesorului 8086, dar *bus*-ul extern de date este pe 8 biţi. El este compatibil software 100% cu 8086, cuprinde toate instrucţiunile acestuia şi are 24 de moduri de adresare: directă, indirectă cu bază şi/sau indexare şi 8 sau 16 biţi pentru deplasament.

Opţiunile de bază, indexare şi deplasament furnizează 3 niveluri de indexare care permit efectuarea cu instrucţiunile de bază a operaţiilor cu şiruri sau matrici de date. 8088 poate adresa pînă la 1 Moctet de memorie, segmentată

în secțiuni de cîte 64 Kocteți. Adresarea memoriei se face folosind unul din registrele de segment ca bază și un deplasament de 16 biți.

Microprocesorul MC6809 poate efectua operații pe 16 biți, deși are *bus*-ul extern de date pe 8 biți. Cu un set de instrucțiuni îmbunătățit față de MC6800 și o arhitectură a registrelor mai puternică, MC6809 prezintă facilități suplimentare pentru implementarea compilatoarelor unor limbaje de nivel înalt.

Pentru realizarea unor sisteme mai compacte, Motorola a pus la punct *microprocesorul MC6802* care are performanțe situate între MC6800 și MC6809. Acesta dispune de un set de instrucțiuni MC6800 îmbunătățit, un oscilator și o memorie RAM.

În paralel cu perfecționarea microprocesoarelor pe 8 biți au apărut și s-au perfecționat microprocesoarele din cea de-a treia categorie — *microcalculatoarele single-chip*, de exemplu 8048, MC6801, MC6805, Z8, COPS etc. (vezi tabelul 3.6).

Microprocesorul 8048 reprezintă o structură tipică de microcalculator într-un singur circuit integrat. Arhitectura internă a acestuia, inspirată din structura simplificată a unui microcalculator realizat cu microprocesorul *8080*, cuprinde următoarele părți:

— un microprocesor tipic de 8 biți;
— o memorie ROM de 1 Koctet;
— o memorie RAM de 64 octeți;
— 3 *port*-uri de I/E;
— un numărător programabil de 8 biți;
— stivă cu 8 niveluri;
— sistem de întreruperi prin hardware cu un nivel.

Microprocesorul MC6801 este o variantă îmbunătățită a microprocesorului MC6800 și mai conține: memorie RAM de 128 octeți, memorie ROM de 2048 octeți, un generator de ceas și 29 linii de I/E. Procesorul are un *port* serial de I/E și un numărător de ceas programabil.

Unul dintre cele mai performante microprocesoare, din categoria microcalculatoarelor *single-chip*, este *Z8*, produs de firma Zilog. Acesta cuprinde: un procesor pe 8 biți cu un set de 47 de instrucțiuni, o memorie ROM de 2048 octeți, o memorie RAM de 124 octeți, două ceasuri/numărătoare și un *port* serial de I/E din cele 32 linii de I/E.

O altă direcție a perfecționării microprocesoarelor o constituie *microprocesoarele specializate* destinate să fie programate de către utilizatori. Astfel de procesoare sînt *S2811*, periferic de prelucrare a semnalelor, produs de firma AMI, *9511* și *9512*, procesoare matematice în virgulă mobilă, produse de firma AMD. De fapt, aceste circuite se situează la granița dintre unitățile de prelucrare și dispozitivele periferice, putînd acționa ca un periferic pentru alt procesor, deși au facilități proprii de prelucrare a informației.

Microprocesorul 2920 produs de Intel este un microcalculator analogic într-un singur circuit integrat. Spre deosebire de alte microcalculatoare pe un singur circuit integrat acest circuit acceptă semnale analogice de intrare și produce la ieșire semnale analogice care au fost prelucrate numeric. Prima versiune a microprocesorului 2920 include o unitate aritmetică logică pe

28 biți, care prelucrează informația digitală, o memorie EPROM de 192×24 biți, o memorie RAM de 40×25 biți și un circuit de ceas. Acest microprocesor are 4 intrări analogice și 8 ieșiri, care pot fi: toate analogice, 4 analogice și 4 digitale, sau toate 8 digitale. Microprocesorul funcționează la frecvența de 2,5 MHz, caracteristica lui esențială fiind aceea că toate instrucțiunile se execută într-un singur ciclu-mașină, adică în 400 ns.

În paralel cu perfecționarea tehnologică a microprocesoarelor s-au dezvoltat și diverse componente software: asambloare, sisteme de operare, compilatoare pentru limbaje de nivel înalt, simulatoare și biblioteci de programe de aplicație.

Pentru a ține pasul, pe cît este posibil, cu evoluția rapidă a microprocesoarelor, proiectanții au căutat ca la fiecare produs nou să se poată utiliza programele de aplicații și alte resurse software deja existente pentru alte microprocesoare. Astfel, microprocesorul 8080 este compatibil software cu 8008, predecesorul său, 8085 este compatibil 100% software cu 8080, 8086 este compatibil 100% software cu 8080/8085 etc.

Pentru a dezvolta și pune la punct atît programele de aplicații, programele generale pentru o anumită familie de microprocesoare, cît și sistemele hardware particulare, sînt necesare sisteme de dezvoltare generale. Astfel, fiecare firmă, după lansarea pe piață a unui tip de microprocesor, a pus la punct și un sistem de dezvoltare adecvat.

Dintre realizările de acest gen amintim: firma Intel — *INTELLEC* pentru 8080, *INTELLEC Series II* pentru 8080, 8085 A și 8048, *INTELLEC Series III* pentru iAPX—86/88 (8080, 8086); Firma Zilog — *MCZ 1/05* pentru Z8, Z80 și Z8000, *PDS 8000* pentru Z8, Z80, Z80 A și Z8000; firma Motorola — *EXORciser* pentru 6800, 6801, 6805, 6809, MC141000 și 3870; *EXORmacs* pentru 68000; firma Tektronix — 8002 A *Microprocessor Development Lab* pentru 8080A, 8085A, 8049, 8035, 8039-6, 8021, 6800, 6809, 9900, Z80A, F8, 3870, 3872, 1802, 8086, 28000, 68000, 6500/1 și Z8000 [29], [30]. Cu ajutorul unor astfel de sisteme de dezvoltare se pun la punct: asamblorul, sistemele de operare, compilatoarele pentru diferitele limbaje de nivel înalt, programele de aplicații, precum și diferitele configurații particulare hardware.

Dezvoltarea tehnologiilor de fabricație și creșterea complexității programelor de lucru au condus la ideea implementării în hardware a anumitor componente software. Astfel, după lansarea microprocesorului 8086, firma Intel a produs două astfel de *coprocesoare*: *8089* — un procesor de I/E și *8087* — procesor de virgulă mobilă. Datorită seturilor de instrucțiuni specializate, aceste două coprocesoare eliberează utilizatorul de sarcina scrierii unor rutine complicate de I/E și de calcul în virgulă mobilă. În continuare Intel va integra diferite părți ale sistemului de operare și apoi ale unor limbaje de nivel înalt, de exemplu PASCAL.

Software-ul complex este și va continua să fie cheia în aplicațiile microprocesoarelor *single-chip*, astfel încît producătorii de microprocesoare își vor îndrepta efortul spre extinderea implementării sistemelor de operare și limbajelor de nivel înalt în hardware.

Firma National Semiconductor a anunțat implementarea limbajului *ADA* într-un singur circuit integrat, iar firma Zilog va implementa într-un circuit integrat compilatorul FØRTRAN, care va fi disponibil diverșilor uti-

lizatori în primăvara anului 1982 [3]. O altă realizare în această direcție este cea a firmei Western Digital care a produs un set de 5 circuite integrate ce constituie o mașină PASCAL. Acest set de circuite integrate reprezintă o unitate de prelucrare pe 16 biți, ce poate executa programe PASCAL în cod P, direct, fără intervenția unui compilator sau interpretor.

Ținînd cont de evoluția rapidă a microprocesoarelor se presupune că în următorii ani proiectanții de sisteme dedicate pentru diferite aplicații trebuie să-și îndrepte atenția spre dezvoltarea software-ului de aplicație, deoarece proiectanții de circuite ne vor oferi pe de-o parte sisteme de operare și limbaje de nivel înalt înglobate într-un singur circuit, iar pe de altă parte, microcalculatoare generale din ce în ce mai performante.

3.2. MICROPROCESOARE PE 8 BIȚI

3.2.1. MICROPROCESORUL 8080

Microprocesorul 8080 este unul dintre cele mai cunoscute microprocesoare pe 8 biți și este fabricat în tehnologie NMOS. A fost produs pentru prima dată de firma Intel în anul 1974 fiind rezultatul perfecționării tehnologiei folosite la fabricarea microprocesorului 8008, produs în anul 1972, în tehnologie PMOS, și apoi a perfecționării structurii acestuia. Microprocesorul 8080 are 78 instrucțiuni (vezi §5.3.) în care sînt incluse și instrucțiunile microprocesorului 8008, asigurîndu-se compatibilitatea de jos în sus.

Caracteristicile generale ale microprocesorului 8080 sînt:

— compatibilitate TTL la intrare și ieșire;

— posibilitatea adresării unei memorii externe de maximum 64 Kocteți;

— posibilitatea adresării a 256 dispozitive de intrare și 256 dispozitive de ieșire;

— mod de lucru în binar și binar-zecimal;

— formează stive de lucru în memoria RAM externă microprocesorului;

— timpul de execuție a unei instrucțiuni care necesită 4 cicli de ceas este de $4 \times 0{,}5 = 2$ μs, iar al unei instrucțiuni care necesită 18 cicli de ceas este $18 \times 0{,}5 = 9$ μs;

— tensiuni de alimentare: +5V, −5 V și +12 V;

— are o singură intrare de întrerupere mascabilă prin program care permite inserarea pe magistrala de date a uneia dintre cele 8 instrucțiuni RESTART de către dispozitivul care a provocat întreruperea.

Arhitectura internă a microprocesorului 8080 este prezentată în figura 5.1.

Procesorul 8080 constă din următoarele unități funcționale:

— zona registrelor de date și adrese;

— unitatea logică-aritmetică, UAL;

— registrul de instrucțiuni și unitatea de comandă și temporizare;

— o magistrală internă de date bidirecțională.

Zona registrelor constă dintr-o zonă de memorie RAM organizată în 6 registre de cîte 16 biți fiecare:

— numărătorul de adrese al programului, PC;

— pointerul stivei, SP;

— 6 registre generale de lucru: B, C, D, E, H și L ce pot fi adresate, ca perechi de registre cu lungimea de 16 biți sau individual, ca registre de 8 biți;
— un registru pereche temporar, notat WZ.

Modul de utilizare a PC, SP și registrelor generale va fi prezentat în capitolul 5. Registrul temporar WZ nu este accesibil programatorului și este folosit de procesor pentru execuția internă a instrucțiunilor.

O informație de un octet poate fi transferată între zona de memorie afectată registrelor și magistrala internă, BUS INTERN DATE-8 BIȚI, utilizînd circuitele pentru selectarea unui registru, SELECȚIE REGISTRU, și MULTIPLEXORUL. Operația de incrementare/decrementare a unui dublu registru este realizată cu ajutorul circuitelor din blocul OPERATOR ± 1 la ieșirea căruia este conectat și registrul de 16 biți servind la păstrarea adresei de memorie trimisă de procesor pe magistrala externă de adrese cu ajutorul circuitelor de comandă conținute în blocul denumit BUFFER ADRESE.

Unitatea aritmetică-logică, UAL, conține un sumator paralel de 8 biți, la care sînt atașate:

— un ACUMULATOR de 8 biți, folosit de utilizator;
— un ACUMULATOR TEMPORAR de 8 biți, care nu este accesibil utilizatorului, fiind folosit pentru memorarea primului operand;
— un REGISTRU TEMPORAR de 8 biți, care nu este accesibil utilizatorului, fiind folosit pentru memorarea celui de-al doilea operand;
— 5 indicatori de condiții (§5.2.2);
— un circuit de corecție pentru lucrul în BCD, CORECȚIE ZECIMALĂ.

UAL este legată de registrul temporar, acumulatorul temporar și registrul indicatorilor de condiții și permite execuția operațiilor aritmetice, logice și de deplasare. Rezultatul unei operații se extrage din UAL pe magistrala internă de date și se introduce în acumulator, indicatorii de condiții fiind poziționați în conformitate cu valoarea rezultatului și cu tipul instrucțiunii curente.

În timpul execuției unei instrucțiuni, codul instrucțiunii este transferat în REGISTRUL DE INSTRUCȚIUNI a cărui ieșire este interpretată cu ajutorul decodificatorului de instrucțiuni. Ieșirea decodificatorului, asociată cu semnalele de ceas, este exploatată de CIRCUITELE DE COMANDĂ ȘI TEMPORIZARE care generează semnalele ce dirijează activitatea internă și externă a procesorului.

Circuitul BUFFER BUS DATE este folosit pentru izolarea magistralei interne de date a procesorului, BUS INTERN DATE, de magistrala externă de date, D_0 ,... ,D_7. Într-o operație de ieșire, conținutul magistralei interne se află în zona tampon de 8 biți de unde mai departe este transmis spre exterior. Într-o operație de intrare, datele de pe magistrala externă sînt transferate în zona tampon a magistralei interne și de aici pe magistrala internă a procesorului.

Amplasarea pinilor microprocesorului 8080 este prezentată în figura 3.1*a* iar semnificația și modul de utilizare al acestora sînt detaliate în §4.2.1 și § 4.2.2.2.

În figura 3.2 este dată arhitectura standard a unui microcalculator 8080, iar în capitolul 4 se prezintă realizarea unui astfel de microcalculator. Arhitectura și *pinout*-ul, conexiunile externe, ale diverselor circuite utilizate

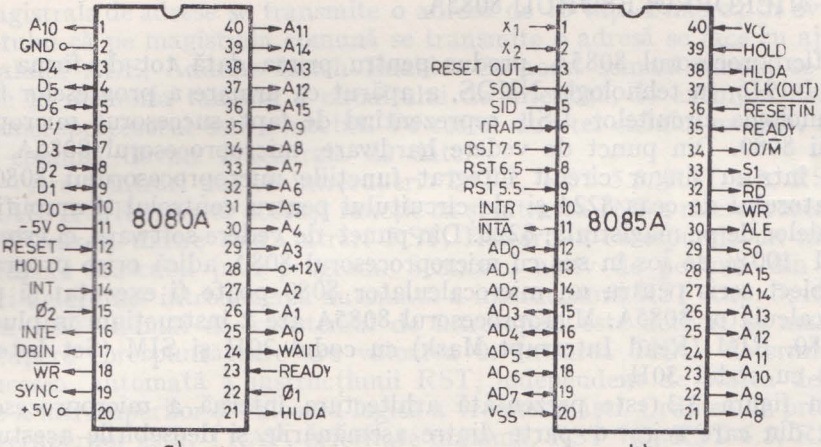


Fig. 3.1. Conexiunile externe ale microprocesoarelor:
a — 8080A; *b* — 8085A

pentru obținerea unui microcalculator sînt descrise în cataloagele firmelor producătoare [9].

Pentru înțelegerea completă a funcționării microprocesorului 8080 informația conținută în acest paragraf trebuie completată cu descrierea semnalelor de interfață cu exteriorul și diagrama de stări a procesorului din capitolul 4, precum și cu descrierea setului de instrucțiuni și modurilor de adresare conținute în capitolul 3.

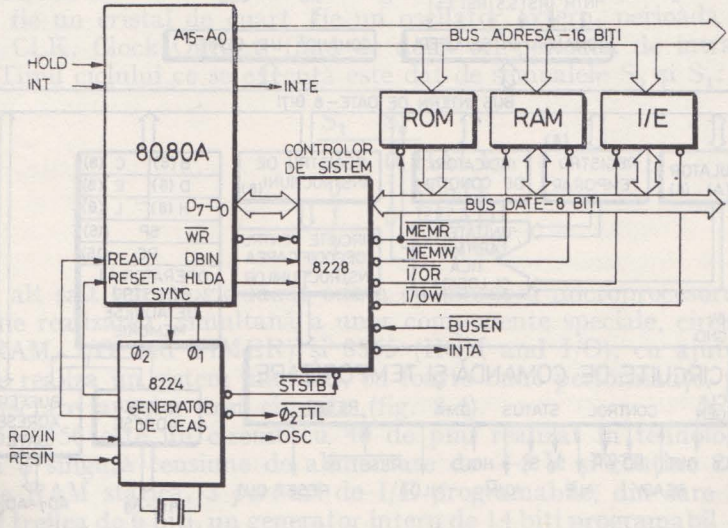


Fig. 3.2. Arhitectura standard a unui microcalculator 8080

3.2.2. MICROPROCESORUL 8085A

Microprocesorul 8085A, produs pentru prima dată tot de firma Intel, în anul 1976, în tehnologia NMOS, a apărut ca urmare a progreselor făcute în tehnologia circuitelor LSI, reprezentînd de fapt succesorul microprocesorului 8080. Din punct de vedere hardware, microprocesorul 8085A înglobează într-un singur circuit integrat funcțiile microprocesorului 8080, ale generatorului de ceas 8224 și ale circuitului pentru controlul și amplificarea semnalelor de pe magistrală, 8228. Din punct de vedere software, el este compatibil 100% de jos în sus cu microprocesorul 8080, adică orice program în cod-obiect scris pentru un microcalculator 8080 poate fi executat și pe un microcalculator 8085A. Microprocesorul 8085A are 2 instrucțiuni în plus față de 8080, RIM (Read Interrupt Mask) cu codul 20H și SIM (Set Interrupt Mask) cu codul 30H.

În figura 3.3 este prezentată arhitectura internă a microprocesorului 8085A din care reies o parte dintre asemănările și deosebirile acestuia cu microprocesorul 8080, iar în figura 3.1 *b* este prezentată amplasarea celor 40 pini ai acestuia.

Îmbunătățirile aduse microprocesorului 8085A, în comparație cu 8080, sînt:

— Necesită o singură tensiune de alimentare, +5 V.

— Frecvența de lucru este de 3 MHz față de 2 MHz la 8080, ciclul unei instrucțiuni fiind de 1,3 μs.

— Pe magistrala comună $AD_0 \div AD_7$ se transmit prin multiplexare în timp atît cei 8 biți ai magistralei de date, cît și cei mai puțin semnificativi 8 biți ai magistralei de adrese. Pe durata primei stări a fiecărui ciclu-mașină

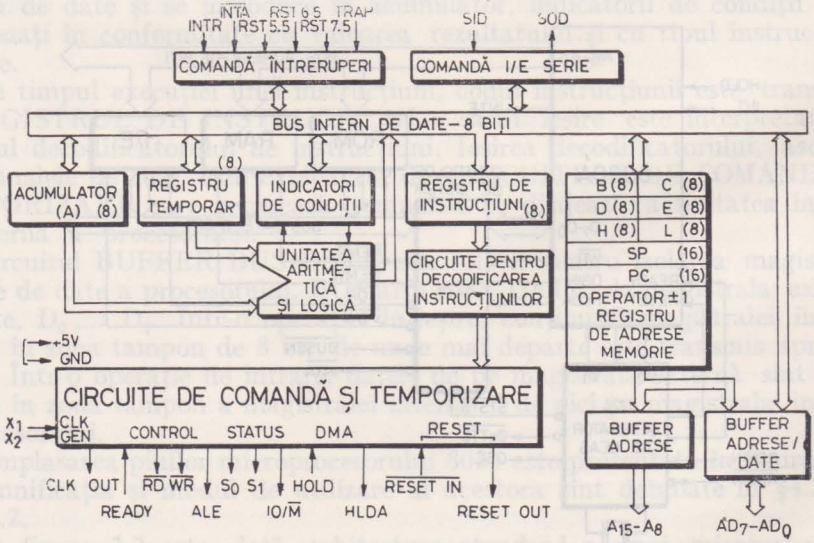


Fig. 3.3. **Arhitectura microprocesorului 8085A**

pe magistrala de adrese se transmite o adresă de 16 biți. Punerea în evidență a faptului că pe magistrala comună se transmite o adresă se face cu ajutorul semnalului ALE, Address Latch Enable. Cu acest semnal adresa se poate înscrie în memoria tampon a circuitului de interfață, de exemplu 8155, cu care microprocesorul este conectat. Pe restul duratei ciclului-mașină, magistrala comună devine magistrală de date.

— Are 5 intrări pentru întreruperi: INTR, RST 5.5, RST 6.5, RST 7.5 și TRAP. INTR execută aceeași funcție ca și intrarea INT a microprocesorului 8080. Fiecăreia dintre cele 3 intrări RST, Restart, îi corespunde un nivel de întrerupere mascabil prin program. Semnalul activ de pe una din aceste intrări determină introducerea automată a instrucțiunii RST (cap. 5, instrucțiunea RST) numai dacă sistemul de întreruperi este activat și masca de întrerupere corespunzătoare are valoarea 0. Intrarea TRAP determină introducerea automată a instrucțiunii RST, independent de starea de autorizare a întreruperilor și valoarea logică a vreunei măști. Ordinea de prioritate a întreruperilor este: TRAP, prioritate maximă, RST 7.5, RST 6.5, RST 5.5 și INTR, prioritate minimă. Vectorii de întrerupere corespunzători intrărilor RST 5.5, RST 6.5, RST 7.5 și TRAP sînt plasați în memorie, respectiv la adresele: 2CH, 34H, 3CH și 24H.

— Liniile seriale de I/E bazate pe intrările SID și SOD sînt prevăzute pentru a fi exploatate prin program: conținutul liniei SID, Serial Input Data line, este încărcat în bitul 7 al acumulatorului la executarea instrucțiunii RIM; ieșirea SOD, Serial Output Data line, este poziționată sau ștearsă de instrucțiunea SIM.

— Controlul vehiculării informației pe magistrale este efectuat cu ajutorul semnalelor $\overline{RD}$, $\overline{WR}$ și IO/$\overline{M}$.

— $\overline{RESET\ IN}$ este intrarea de ștergere generală.

— Circuitul intern de ceas este legat la intrările X_1 și X_2 unde se poate conecta fie un cristal de cuarț, fie un oscilator extern, perioada semnalului de ceas CLK, Clock Output, fiind de două ori perioada de intrare.

— Tipul ciclului ce se execută este dat de semnalele S_0 și S_1:

S_1	S_0	
0	0	HALT
0	1	WAIT
1	0	READ
1	1	FETCH

Un alt salt tehnologic făcut odată cu lansarea microprocesorului 8085 îl constituie realizarea simultană a unor componente speciale, circuitele 8155/8156 (RAM, I/O and TIMER) și 8355 (ROM and I/O), cu ajutorul cărora se poate realiza un sistem minimal, cu foarte bune performanțe, prin simpla interconectare a celor trei circuite (fig. 3.4).

8155/8156 este un circuit cu 40 de pini realizat în tehnologie NMOS, necesită o singură tensiune de alimentare de +5 V și conține 2048 biți de memorie RAM statică, 3 *port*-uri de I/E programabile, din care 2 de cîte 8 biți și al treilea de 6 biți, un generator intern de 14 biți programabil — TIMER, o magistrală multiplexată de adrese și date și un buffer intern pentru memo-

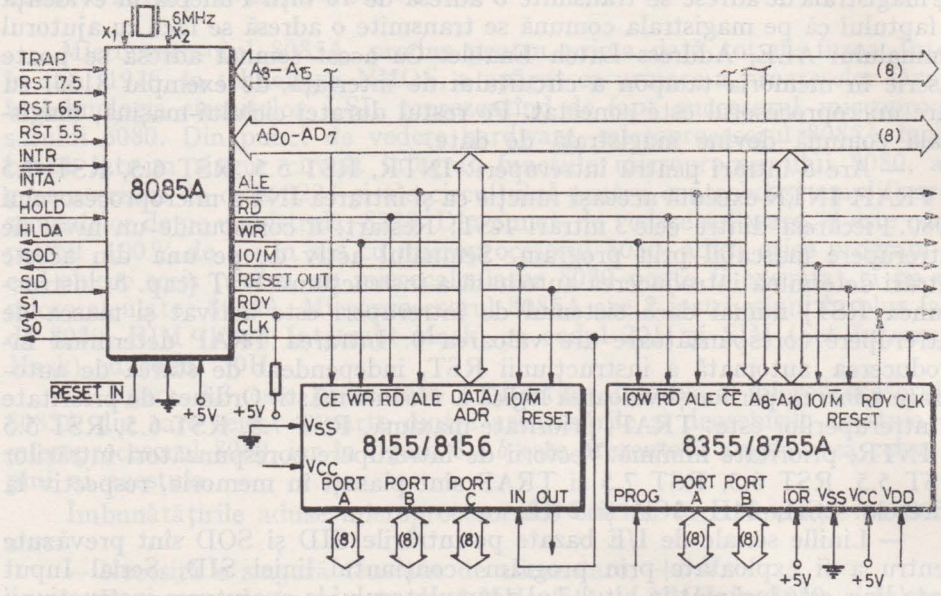


Fig. 3.4. Arhitectura unui microsistem minimal 8085A

rarea adresei. Deosebirea dintre cele două circuite este că la 8155 semnalul CE, Chip Enable, este activ jos în timp ce la 8156 acest semnal este activ sus.

8355 este un circuit cu 40 de pini, necesită o singură tensiune de alimentare de +5 V și conține: o memorie PROM de 2 Kocteți, 2 *port*-uri de I/E de cîte 8 biți, o magistrală multiplexată de adrese și date și un buffer intern pentru memorarea adresei. Fiecare din cele 2 *port*-uri de I/E sînt programabile individual ca intrare sau ca ieșire. Un alt circuit cu care poate fi înlocuit 8355 este 8755A care se deosebește de acesta prin faptul că memoria înglobată în el este reprogramabilă.

Seria 8085 este completată cu circuite de interfață echivalente celor din seria microprocesorului 8080 [9], [14]:

8251A — Programmable Communications Interface;

8253-5 — Programmable Interval Timer;

8255A-5 — Programmable Peripheral Interface;

8257-5 — Programmable DMA Controller;

8259-5 — Programmable Interrupt Controller.

Pentru interfațarea cu componente standard de memorie așa cum sînt circuitele Intel 8102A, 8101A, 8111A, 8316A, 8308, 2104 și 2116 este necesară demultiplexarea magistralei de adrese/date. Acest lucru se poate realiza cu ajutorul unui circuit Intel 8212. În acest fel întreaga magistrală de adrese, 16 biți, devine disponibilă interfeței standard a componentelor de memorie, ca în figura 3.5, unde este prezentată arhitectura unui asemenea microsistem.

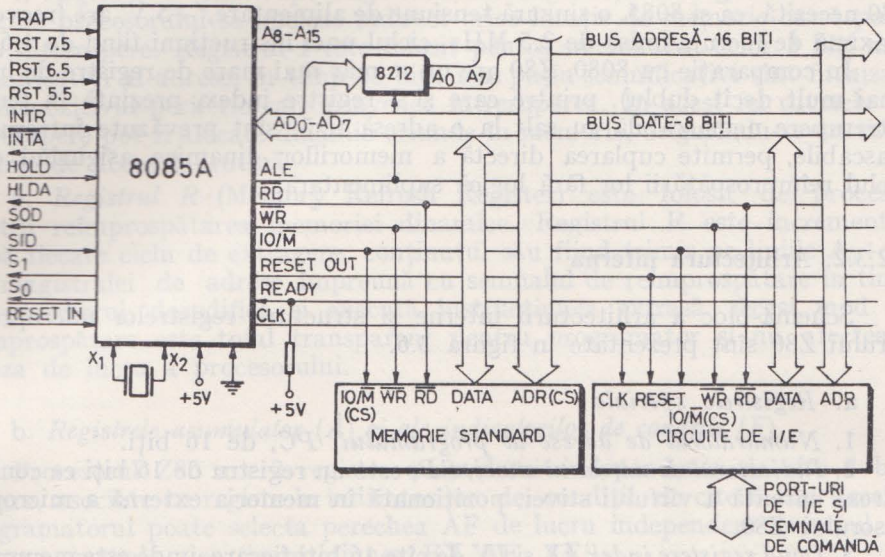


Fig. 3.5. Arhitectura generală a unui microsistem 8085A

3.2.3. MICROPROCESORUL Z80

3.2.3.1. Generalități

Microprocesorul Z80 este produs de firma Zilog în tehnologie NMOS, într-un singur *chip* cu 40 conexiuni exterioare. Acest microprocesor a fost realizat de cîțiva proiectanți care au participat și la proiectarea microprocesorului Intel 8080. El înglobează caracteristicile microprocesorului 8080, ale circuitului de comandă și de amplificare a semnalelor de pe magistrale, 8228, ale generatorului de ceas, 8224, avînd în plus și alte facilități ce vor fi descrise în continuare.

Z80 are 158 instrucțiuni în care sînt incluse și instrucțiunile lui 8080 și 8085A, fapt ce permite utilizarea programelor deja scrise pentru unul din sistemele amintite pe un microsistem Z80. Codurile celor 78 instrucțiuni ale microprocesorului 8080 sînt cuprinse în instrucțiunile microprocesorului Z80, așa că un program-obiect rezultat pe 8080 poate fi executat pe un microcalculator Z80. Noile instrucțiuni includ: operații suplimentare pe 4, 8 și 16 biți cu mai multe moduri de adresare, transferul și verificarea conținutului unor zone de memorie, manipularea și testarea informațiilor de un bit din orice registru sau locație de memorie, extinderea operațiilor aritmetice pe 16 biți, extinderea aritmeticii în cod BCD, mai multe instrucțiuni de I/E, o gamă mai largă de operații de deplasare și rotație (aritmetice și logice) a conținutului unei locații de memorie sau registru.

Toate aceste facilități permit rezolvarea unei aplicații într-un timp mai scurt și reducerea spațiului memoriei ocupate (între 25% și 50%) pe un sistem realizat cu Z80 în comparație cu un sistem similar bazat pe 8080.

Z80 necesită, ca și 8085, o singură tensiune de alimentare, +5 V, are frecvența maximă de lucru admisă de 2,5 MHz, ciclul unei instrucțiuni fiind de 1,6 μs.

În comparație cu 8080, Z80 are un număr mai mare de registre de lucru (mai mult decît dublu), printre care și 2 registre index, prezintă în plus o întrerupere nemascabilă cu salt la o adresă fixă, sînt prevăzute întreruperi mascabile, permite cuplarea directă a memoriilor dinamice asigurînd controlul reîmprospătării lor fără logică suplimentară.

3.2.3.2. Arhitectura internă

Schema-bloc a arhitecturii interne și structura registrelor microprocesorului Z80 sînt prezentate în figura 3.6.

a. *Registrele speciale*

1. *Numărătorul de adrese al programului, PC*, de 16 biți.
2. *Indicatorul de adrese al stivei, SP*, este un registru de 16 biți ce conține adresa curentă a vîrfului stivei, poziționată în memoria externă a microprocesorului Z80.
3. *Două registere index IX* și *IY* de cîte 16 biți fiecare, unde este memorată adresa de bază la adresarea indexată.

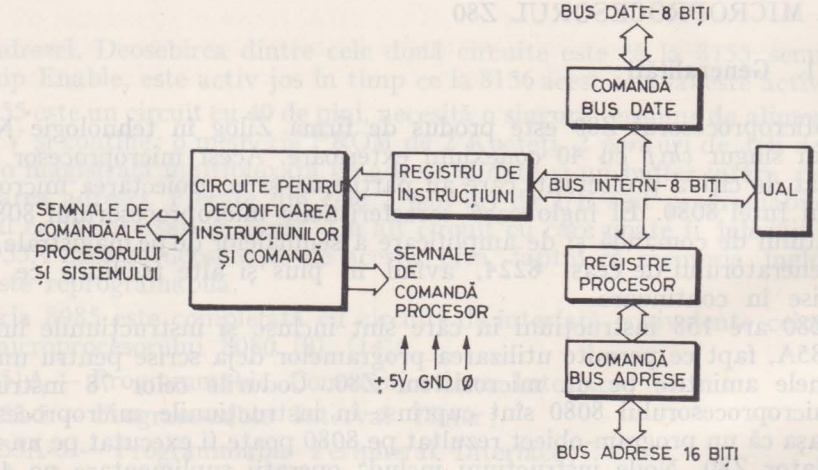


Fig. 3.6. Schema-bloc a microprocesorului Z80 (a); registrele generale accesibile programatorului (b)

4. *Registrul I* (Interrupt Page Address Register) cu lungimea de 8 biţi permite procesorului să apeleze indirect orice locaţie de memorie ca răspuns la o întrerupere. Registrul I este folosit pentru memorarea celor 8 biţi mai semnificativi ai adresei în timp ce biţii mai puţin semnificativi sînt furnizaţi de dispozitivul periferic care provoacă întreruperea. În acest fel rutinele de întrerupere pot fi alocate dinamic oriunde în memorie, asigurîndu-se un timp minim de acces la rutină.

5. *Registrul R* (Memory Refresh Register) este folosit de procesor pentru reîmprospătarea memoriei dinamice. Registrul R este incrementat după fiecare ciclu de extragere, conţinutul său fiind trimis pe liniile $A_0 \div A_6$ ale magistralei de adrese împreună cu semnalul de reîmprospătare în timp ce procesorul decodifică şi execută instrucţiunea extrasă. Acest mod de reîmprospătare este total transparent pentru programator şi nu afectează viteza de lucru a procesorului.

b. *Registrele acumulator (A) şi ale indicatorilor de condiţii (F)*

Procesorul Z80 are 2 registre acumulator independente, de cîte 8 biţi fiecare, asociate cu registrele indicatorilor de condiţii, de cîte 8 biţi fiecare. Programatorul poate selecta perechea AF de lucru independent de setul de registre generale ales (instrucţiunea EX AF, AF').

Indicatorii de condiţii. Fiecare din cele 2 registre F conţine 6 biţi de informaţie care sînt poziţionaţi în conformitate cu rezultatul operaţiei efectuate de UAL. Patru din aceşti biţi sînt testabili prin program şi sînt folosiţi de instrucţiunile condiţionate de salt, apel şi revenire. Aceştia sînt: indicatorul de transport-C (CARRY), indicatorul de rezultat nul-Z (ZERO), indicatorul de semn-S (SIGN), indicatorul de paritate/depăşire-P/V (PARITY/OVERFLOW). Indicatorul P/V este utilizat pentru a indica paritatea rezultatului din acumulator sau depăşirea aritmetică în cazul operaţiilor în complement faţă de 2.

Ceilalţi 2 biţi de informaţie din registrul F sînt folosiţi pentru corecţia operaţiilor de adunare şi scădere în aritmetica numerelor în cod BCD. Ei nu sînt accesibili programatorului. Aceştia sînt:

1. Indicatorul de transport auxiliar-H (HALF-CARRY) indică un transport în cazul unei operaţii de adunare sau un împrumut în cazul unei operaţii de scădere în/din bitul 4.

2. Indicatorul de adunare/scădere-N arată tipul instrucţiunii executate anterior operaţiei de corecţie, DAA. Introducerea acestui indicator a fost necesară deoarece algoritmul de corecţie a rezultatului unei operaţii de adunare este diferit de cel al operaţiei de scădere a 2 numere în cod BCD.

Registrul indicatorilor de condiţii este accesibil programatorului şi are următoarea structură:

S	Z	X*	H	X*	P/V	N	C
7							0

* X arată că valoarea bitului corespunzător este nedeterminată.

c. *Registrele generale de lucru*

În grupul registrelor de 8 biți sînt 2 seturi de cîte 6 registre ce pot fi folosite individual, ca registre de 8 biți sau ca registre duble de 16 biți. Un set conține registrele B, C, D, E, H și L. Setul complementar cuprinde registrele: B', C', D', E', H' și L'. Programatorul poate selecta unul din cele 2 seturi de registre printr-o comandă de schimb adecvată (instrucțiunea EXX).

d. *Unitatea aritmetică-logică (UAL).*

UAL execută instrucțiunile aritmetice și logice pe 8 biți ale procesorului: adunare, scădere, ȘI logic (AND), SAU logic (OR), SAU EXCLUSIV logic (XOR), comparare, deplasări și rotații (aritmetice și logice) spre stînga sau dreapta, incrementare, decrementare, poziționare/ștergere bit și testare bit.

e. *Registrul de instrucțiuni și circuitele de decodificare* furnizează semnalele de comandă necesare funcționării procesorului.

3.2.3.3. Descrierea semnalelor externe

În figura 3.7 sînt prezentate cele 40 de conexiuni exterioare ale microprocesorului Z80. Semnificația și funcțiile acestora sînt descrise în continuare.

$A_0 \div A_{15}$, magistrala de adrese de 16 biți are ieșirea *tri-state* și este activă sus. Aceasta poate adresa o memorie externă de maximum 64 Kocteți și dispozitivele periferice de I/E.

$D_0 \div D_7$, magistrala de date are lungimea de 8 biți, este bidirecțională, are intrări/ieșiri active sus de tip *tri-state*. Aceasta permite schimbul de informații dintre procesor și memoria externă, respectiv dintre procesor și dispozitivele de I/E.

$\overline{M_1}$ este o ieșire activă jos care marchează întotdeauna primul ciclu al unei instrucțiuni. $\overline{M_1}$ apare în ciclurile de extragere a codului instrucțiunii ce urmează a fi executată sau în codificarea cererilor de întrerupere împreună cu $\overline{IORQ}$ activ.

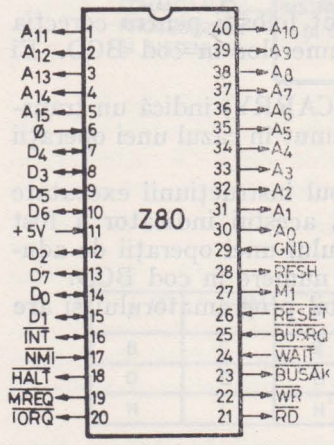


Fig. 3.7. Conexiunile externe ale microprocesorului Z80

$\overline{MREQ}$ (Memory Request) este un semnal *tri-state* activ jos care arată că magistrala de adrese conține o adresă validă pentru operația de citire sau scriere din/în memorie.

$\overline{IORQ}$ (Input/Output Request) este un semnal *tri-state* activ jos care arată că pe magistrala de adrese primii 8 biți, $A_0 \div A_7$, conțin adresa unui periferic de I/E necesară pentru execuția unei operații de citire sau scriere; împreună cu semnalul $\overline{M_1}$ activ confirmă o cerere de întrerupere.

$\overline{RD}$ (Memory Read) este un semnal *tri-state* activ jos ce apare în ciclurile în care se efectuează o citire din memorie sau de la un dispozitiv periferic.

$\overline{WR}$ (Memory Write) este un semnal *tri-state* activ jos și apare în ciclurile de scriere în memorie sau pe un dispozitiv periferic.

$\overline{RFSH}$ (Refresh) este un semnal activ jos care arată că cei 7 biți mai puțin semnificativi, $A_0 \div A_6$, ai magistralei de adrese, conțin o adresă de reîmprospătare a memoriei dinamice.

$\overline{HALT}$ (Halt state) semnalează exteriorului că procesorul s-a blocat în urma executării unei instrucțiuni HALT. Deblocarea procesorului se poate face fie printr-o întrerupere nemascabilă, fie printr-o întrerupere mascabilă, dacă ea poate fi acceptată de procesor sau prin inițializarea generală a procesorului.

$\overline{WAIT}$ este un semnal de intrare activ jos ce arată procesorului că memoria sau dispozitivele de I/E nu sînt pregătite pentru un transfer de date. Acest semnal permite sincronizarea dintre procesor și memoria externă sau un dispozitiv de I/E.

$\overline{INT}$ (Interrupt Request) este semnalul de cerere de întrerupere mascabilă din partea dispozitivelor de I/E, fiind activ jos. O întrerupere este onorată după executarea instrucțiunii curente, dacă sistemul de întreruperi este activ și $\overline{BUSRQ}$ nu este activ. Cînd procesorul acceptă întreruperea, un semnal de recunoaștere, $\overline{IORQ}$, este emis spre exterior la începutul ciclului următoarei instrucțiuni.

$\overline{NMI}$ (Non Maskable Interrupt) este o intrare activă pe frontul căzător și semnalează apariția unei întreruperi nemascabile. Cererea de întrerupere nemascabilă are prioritate mai mare decît $\overline{INT}$, fiind întotdeauna recunoscută la sfîrșitul execuției instrucțiunii curente. $\overline{NMI}$ forțează automat execuția programului de la adresa 66H. PC este salvat în stivă, așa că după execuția programului ce tratează această întrerupere se poate reveni în programul întrerupt.

$\overline{RESET}$ este o intrare activă jos. $\overline{RESET}$ forțează numărătorul de adrese al programului în zero și inițializează procesorul. Inițializarea procesorului include următoarele operații:

1. dezactivează bistabilul de întrerupere IFF;
2. încarcă zero în registrele I și R;
3. activează modul 0 de întrerupere.

$\overline{BUSRQ}$ (Bus Request) este o intrare activă jos. Semnalul de *cerere bus*, $\overline{BUSRQ}$, este folosit pentru trecerea magistralei de adrese, magistralei de date și a semnalelor de comandă, care sînt de tip *tri-state*, în starea de impedanță înaltă, astfel încît și alte dispozitive să poată controla aceste magistrale. Cînd $\overline{BUSRQ}$ este activat, procesorul va poziționa magistrala într-o stare de impedanță înaltă de îndată ce ciclul-mașină curent a fost terminat.

$\overline{BUSAK}$ (Bus Acknowledge) este o ieșire activă jos. Semnalul de *achitare cerere bus* indică dispozitivului periferic solicitant că magistrala de adrese, magistrala de date și semnalele de comandă cu ieșirea *tri-state* au fost puse în starea lor de impedanță înaltă și că dispozitivul periferic poate acum controla aceste semnale.

Ø este un semnal de ceas în nivel TTL.

3.2.3.4. Comportarea în întreruperi

Microprocesorul Z80 are 2 tipuri de întreruperi: o întrerupere mascabilă prin program (INT) și o întrerupere nemascabilă (NMI).

Întreruperea nemascabilă nu poate fi dezactivată prin program și va fi acceptată în orice moment de către procesor. Ea este în general rezervată pentru funcția cea mai importantă dintr-o aplicație deservită de un microcalculator Z80, care trebuie rezolvată fără nici o întîrziere.

Întreruperea mascabilă poate fi selectiv activată sau dezactivată prin program cu ajutorul instrucțiunilor EI, respectiv DI. Acest lucru permite programatorului să dezactiveze sistemul de întreruperi pe o anumită porțiune a unui program, unde acesta nu permite a fi întrerupt.

Procesorul Z80 are 2 bistabili, IFF_1 și IFF_2, care sînt poziționați după execuția instrucțiunii EI și șterși după execuția instrucțiunii DI sau achitarea unei întreruperi prin execuția instrucțiunii RETI.

De fapt, bistabilul IFF_2 este utilizat pentru memorarea stării bistabilului IFF_1 în momentul în care apare o întrerupere nemascabilă de prioritate absolută, care șterge bistabilul IFF_1. Starea bistabilului IFF_1 este refăcută după achitarea întreruperii nemascabile, adică după execuția instrucțiunii RETN.

Procesorul poate fi programat în așa fel încît poate să răspundă la oricare din cele 3 moduri posibile.

MOD 0. Acesta este identic cu modul de răspuns în întreruperi al microprocesorului 8080A. Unitatea care solicită întreruperea generează către procesor o instrucțiune RESTART care va da controlul programului corespunzător de tratare a întreruperii.

MOD 1. În acest mod de lucru, acceptarea unei întreruperi de către procesor produce saltul la adresa 38H, echivalent cu instrucțiunea RST 7 a microprocesorului 8080A.

MOD 2. Este modul cel mai frecvent de tratare a întreruperilor generate de diversele periferice cuplate la un microsistem Z80. În acest mod este necesară stabilirea anterioară de către programator a unui tabel cu adresele rutinelor pentru tratarea întreruperilor. În momentul în care se acceptă o întrerupere este format un indicator de 16 biți ce conține adresa din tabelul unde se găsește adresa de start a rutinei corespunzătoare. Registrul I conține octetul de pondere superioară al adresei, ceilalți 8 biți fiind furnizați de perifericul care provoacă întreruperea. De fapt numai 7 biți sînt ceruți de la perifericul care cere întreruperea, deoarece ultimul bit, cel mai puțin semnificativ, este zero. Acest lucru impune ca tabelul de salt cu adresele rutinelor de tratare a întreruperilor să înceapă de la o adresă cu soț.

3.2.3.5. Moduri de adresare

Diversitatea modurilor de calcul al adresei datelor în instrucțiunile de prelucrare sau al adresei-destinație în cazul instrucțiunilor de salt constituie una dintre cele mai importante caracteristici de utilizare ale unui procesor.

În continuare se prezintă modurile de adresare pe care microprocesorul Z80 le pune la dispoziția programatorului.

Adresarea imediată se referă la instrucțiunile care conțin operandul în următorul octet al codului generalizat al instrucțiunii. Exemplu: încarcă în acumulator o constantă.

Adresarea imediată extinsă se referă la instrucțiunile care conțin operandul în octeții doi și trei ai instrucțiunii, octeții următori codului generalizat al instrucțiunii. Exemplu: încarcă un registru de 16 biți cu o constantă.

Adresarea directă în pagina zero. Z80 are o instrucțiune specială de un octet, RST, cu ajutorul căreia se pot apela 8 locații diferite de memorie din pagina zero. Adresa efectivă a acestor locații este dată de biții 5, 4 și 3 ai codului instrucțiunii. Această instrucțiune permite apelul unei subrutine din pagina zero cu ajutorul unui singur octet, deci față de instrucțiunile clasice de apel al unei subrutine se face o economie de memorie de 2 octeți.

Adresarea relativă folosește cel de-al doilea octet al instrucțiunii pentru calculul adresei de salt. Acest număr, denumit și *deplasament*, este interpretat ca număr cu semn în complement față de 2; pentru calculul adresei de salt se adună deplasamentul cu valoarea curentă a numărătorului de adrese care în acest moment indică adresa instrucțiunii ce urmează instrucțiunea de salt (PC + 2). Gama adreselor disponibile este PC — 126, PC + 129, unde PC reprezintă adresa instrucțiunii de salt.

Adresarea extinsă apare în instrucțiunile care conțin adresa locației de memorie referite în octeții 2 și 3 ai acesteia; octetul 2 conține biții mai puțin semnificativi, iar octetul celălalt biții mai semnificativi.

Adresarea indexată constă în adunarea conținutului celui de-al treilea octet al instrucțiunii, *deplasamentul*, cu conținutul unuia din cele 2 registre index, IX sau IY, pentru a obține adresa unei locații de memorie. Codul generalizat al acestor instrucțiuni ocupă 2 octeți și precizează care din cele 2 registre index este folosit. Calculul adresei nu afectează conținutul registrului index utilizat. Deplasamentul e este un număr cu semn în complement față de 2. Adresarea indexată simplifică programele care folosesc tabele de date, deoarece un registru index poate indica adresa de început a tabelului. Sînt necesare 2 registre index, întrucît majoritatea aplicațiilor utilizează mai multe tabele de date. Adresarea indexată se notează simbolic (IX + e) sau (IY + e) în cîmpul de operand.

Adresarea registrelor se referă la instrucțiunile al căror cod de operație specifică unul sau două registre generale ce intervin în operația de prelucrare indicată de instrucțiune. Exemplu: instrucțiunile de transfer al informației între registre conțin adresa registrului-sursă și respectiv destinație.

Adresarea implicită se referă la instrucțiunile al căror cod generalizat implică unul sau mai multe registre ce conțin operanzi. Un exemplu este utilizarea implicită a acumulatorului în operațiile aritmetice și logice ca prim operand.

Adresarea indirectă prin registre folosește registrele pereche pentru adresarea oricărei locații de memorie a unui microcalculator Z80. Un exemplu tipic al acestui mod de adresare este încărcarea unui registru cu informația conținută în locația de memorie a cărei adresă este dată de conținutul perechii de registre HL. Cele mai puternice instrucțiuni ale lui Z80 sînt instrucțiunile

de transfer al unei zone de memorie într-o altă zonă de memorie sau de căutare a unei valori într-un tabel de date situat la adrese succesive. În cazul instrucțiunii de transfer al unei zone de memorie, HL conține adresa zonei sursă, DE — adresa zonei destinație și BC—lungimea ambelor zone.

Adresarea pe bit se referă la instrucțiuni care poziționează, șterg sau testează valoarea unui bit dintr-o locație de memorie sau dintr-un registru de lucru al procesorului. Octetul implicat poate fi referit prin adresare de registru, indirect utilizînd un registru pereche sau indexat. În codul instrucțiunii 3 biți sînt rezervați pentru a arăta care din cei 8 biți ai săi trebuie prelucrat.

Adresarea combinată. O parte din instrucțiuni au mai mult de un operand, ca de exemplu instrucțiunile aritmetice sau de transfer. În acest caz două tipuri de adresare pot fi combinate. De exemplu, încărcarea poate folosi adresarea imediată pentru sursă și indirectă sau indexată pentru destinație.

3.2.3.6. Setul de instrucțiuni

Microprocesorul Z80 are 158 instrucțiuni cu lungime variabilă, ce poate fi de 1, 2, 3 sau 4 octeți, iar codul-mașină al acestora poate avea lungimea de 1 sau 2 octeți. Compatibilitatea de cod cu 8080 este asigurată prin suprapunerea codurilor de operație echivalente ale lui Z80 cu cele 244 de coduri ale lui 8080. Cele 12 coduri rămase disponibile, reprezentînd coduri de operație inexistente pentru 8080, sînt utilizate pentru implementarea celor 80 de instrucțiuni suplimentare ale lui Z80. Din acest motiv în cazul unora dintre instrucțiuni a fost necesară folosirea unor coduri de operație pe doi octeți. În continuare este făcută o prezentare sumară a setului de instrucțiuni Z80, în care instrucțiunile sînt împărțite în mai multe grupe după funcțiile pe care le îndeplinesc.

NOTAȚII FOLOSITE:

b	— indicele unui bit dintr-un octet oarecare conținut într-un registru sau locație de memorie;
cc	— cod de condiție: NZ (Non Zero), Z (Zero), NC (Non Carry), C (Carry), PO (Parity Odd or No Overflow), PE (Parity Even or Overflow), P (Positive), M (Negative, Minus);
d	— destinație reprezentată de un registru sau de o locație de memorie de 8 biți;
dd	— destinație reprezentată de un registru dublu sau de două locații succesive de memorie (16 biți);
e	— deplasament reprezentat pe 8 biți în complement față de 2;
L	— cele 8 locații speciale de apel subrutină din pagina zero situate la adresele: 0, 8, 16, 24, 32, 40, 48, 56;
n	— un număr oarecare pe 8 biți;
nn	— un număr oarecare pe 16 biți;
r	— un registru general de 8 biți: A, B, C, D, E, H, L;
rr	— un registru dublu: AF, BC, DE, HL;
s	— sursă reprezentată de un registru sau de o locație de memorie de 8 biți;
ss	— sursă reprezentată de un registru dublu sau două locații succesive de memorie (16 biți);

s_b — valoarea bitului indice b dintr-o sursă de informație s de 8 biți;
indice L — partea cea mai puțin semnificativă a unui registru dublu;
indice H — partea cea mai semnificativă a unui registru dublu;
( ) — conținutul adresei ce se află între paranteze va fi utilizat ca adresă memorie sau adresă de I/E.

Registrele de 8 biți sînt denumite: A, B, C, D, E, H, L, I și R, iar registrele de 16 biți sînt: AF, BC, DE, HL, SP, PC, IX și IY.

1. *Transfer pe 8 biți*

Mnemonic	Descriere	Comentarii
LD *r, s*	$r \leftarrow s$	$s = r, n$, (HL), (IX+e), (IY+e)
LD *d, r*	$d \leftarrow r$	d = (HL), r, (IX+e), (IY+e)
LD *d, n*	$d \leftarrow n$	d = (HL), (IX+e), (IY+e)
LD A, *s*	$A \leftarrow s$	s=(BC), (DE), (nn), I, R
LD *d*, A	$d \leftarrow A$	d=(BC), (DE), (nn), I, R

2. *Transfer pe 16 biți*

LD *dd, nn*	$dd \leftarrow nn$	dd = BC, DE, HL, SP, IX, IY
LD *dd*, (*nn*)	$dd \leftarrow (nn)$	dd=BC,DE,HL, SP, IX, IY
LD (*nn*), *ss*	$(nn) \leftarrow ss$	ss=BC,DE,HL, SP, IX, IY
LD SP, *ss*	$SP \leftarrow ss$	ss=HL, IX, IY
PUSH *ss*	$(SP-1) \leftarrow ss_H$, $(SP-2) \leftarrow ss_L$	ss=BC,DE,HL, AF, IX, IY
POP *dd*	$dd_L \leftarrow (SP)$ $dd_H \leftarrow (SP+1)$	dd = BC, DE, HL, AF, IX, IY

3. *Schimburi între registre*

EX DE, HL	$DE \leftrightarrow HL$	
EX AF, AF′	$AF \leftrightarrow AF'$	
EXX	$\begin{pmatrix} BC \\ DE \\ HL \end{pmatrix} \leftrightarrow \begin{pmatrix} BC' \\ DE' \\ HL' \end{pmatrix}$	
EX (SP), *ss*	$(SP) \leftrightarrow ss_L$, $(SP+1) \leftrightarrow ss_H$	ss=HL, IX, IY

4. *Transfer al unei zone de memorie*

Mnemonic	Descriere	Comentarii
LDI	(DE) ← (HL) DE←DE+1,HL← HL+1,BC←BC−1	
LDIR	(DE) ← (HL), DE←DE+1,HL← HL+1,BC←BC−1	Se repetă pînă cînd BC = 0
LDD	(DE) ← (HL), DE←DE−1,HL← HL−1,BC←BC−1	
LDDR	(DE) ← (HL), DE←DE−1,HL← HL−1,BC←BC−1	Se repetă pînă cînd BC = 0

5. *Căutare într-o zonă de memorie*

CPI	A−(HL),HL← HL+1,BC←BC−1	
CPIR	A−(HL),HL← HL+1,BC←BC−1	Se repetă pînă BC=0
CPD	A−(HL),HL← HL−1,BC←BC−1	
CPDR	A−(HL),HL← HL−1,BC←BC−1	Se repetă pînă BC=0

6. *Operații aritmetice și logice pe 8 biți*

ADD *s*	A←A+*s*	
ADC *s*	A ← A+*s*+CY	
SUB *s*	A ← A − *s*	
SBC *s*	A ← A−*s*−CY	*s* = *r*, *n*, (HL), (IX+*e*), (IY+*e*)
AND *s*	A ← A ∧ *s*	
OR *s*	A ← A ∨ *s*	
XOR *s*	A ← A ⊕ *s*	
CP *s*	A − *s*	
INC *d*	*d* ← *d* + 1	*d*=*r*, (HL), (IX+*e*), (IY+*e*)
DEC *d*	*d* ← *d* − 1	

7. *Operații aritmetice pe 16 biți*

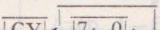
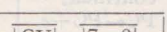
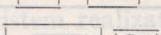
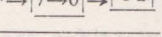
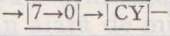
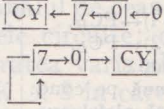
ADD HL, *ss*	HL←HL+*ss*	*ss*=BC, DE, HL, SP
ADC HL, *ss*	HL←HL+*ss*+CY	
SBC HL, *ss*	HL←HL−*ss*−CY	
ADD IX, *ss*	IX←IX+*ss*	*ss*=BC, DE, IX, SP

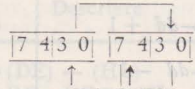
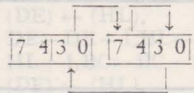
Mnemonic	Descriere	Comentarii
ADD IY, *ss*	IY←IY+*ss*	*ss*=BC, DE, IY, SP
INC *dd*	*dd* ←*dd* +1	*dd*=BC, DE, HL, SP, IX, IY
DEC *dd*	*dd*←*dd* −1	*dd*=BC, DE, HL, SP, IX, IY

8. *Operații speciale*

DAA	Convertește (A) în BCD împachetat	
CPL	A ← $\bar{A}$	
NEG	A ← $\bar{A}$+1	
CCF	CY ← $\overline{CY}$	
SCF	CY ← 1	
NOP		
HALT	Halt UC	
DI	Dezactivează întreruperile	
EI	Activează întreruperile	
IM0	Întreruperi în MOD 0	
IM1	Întreruperi în MOD 1	
IM2	Întreruperi în MOD 2	
RST L	(SP−1)←PC_H, (SP−2)←PC_L, PC_H←0, PC_L←L	

9. *Rotații și deplasări*

RLC *s*		*s* = *r*, (HL),
RL *s*		(IX+*e*),
RRC *s*		(IY+*e*)
RR *s*		
SLA *s*		
SRA *s*		
SRL *s*		

Mnemonic	Descriere	Comentarii
RLD		
RRD		

10. *Prelucrări pe bit*

BIT *b, s* **SET** *b, s* **RES** *b, s*	$Z \leftarrow \bar{s}_b$ $s_b \leftarrow 1$ $s_b \leftarrow 0$	$s = r$, (HL), (IX+*e*), (IY+*e*)

11. *Salturi*

JP *nn* **JP** *cc, nn*	PC ← *nn* Salt pe cond. *cc* adevărată PC←nn, altfel continuă.	*cc* { NZ PO; Z PE; NC P; C M }
JR *e*	PC←PC+*e*	
JR *kk, e*	Salt pe cond. *kk* adevărată PC←PC+*e*, altfel continuă	*kk* { NZ NC; Z C }
JP (*ss*)	PC←ss	*ss*=HL, IX, IY
DJNZ *e*	B ← B−1, dacă B=0 continuă, altfel PC←PC+*e*	

12. *Apeluri subrutine*

CALL *nn*	(SP−1)←PC_H, (SP−2)←PC_L, PC←nn
CALL *cc, nn*	Continuă pe cond. *cc* falsă altfel execută CALL *nn*

13. *Revenire din subrutine și întreruperi*

Mnemonic	Descriere	Comentarii
RET	$PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$	
RET *cc*	Continuă pe cond. *cc* falsă, altfel execută RET	
RETI	Revenire dintr-o întrerupere mas.	
RETN	Revenire dintr-o întrerupere nemas.	

14. *Operații de intrare/ieșire*

IN A, (*n*)	A ← (*n*)	
IN *r*, (C)	*r* ← (C)	
INI	(HL)←(C),HL←HL+1,B←B−1	
INIR	(HL)←(C),HL←HL+1,B←B−1	Continuă pînă B = 0
IND	(HL)←(C),HL←HL−1,B←B−1	
INDR	(HL)←(C),HL←HL−1,B←B−1	Continuă pînă B=0
OUT (*n*),A	(*n*)←A	
OUT (C), *r*	(C)←*r*	
OUTI	(C)←(HL),HL←HL+1,B←B−1	
OTIR	(C)←(HL),HL←HL+1,B←B−1	Continuă pînă B=0
OUTD	(C)←(HL),HL←HL−1,B←B−1	
OTDR	(C)←(HL),HL←HL−1,B←B−1	Continuă pînă B=0

3.2.3.7. Structura unui microsistem realizat cu microprocesorul Z80

În figura 3.8 este prezentată arhitectura unui microsistem simplu realizat cu microprocesorul Z80. În afară de microprocesor, care este conectat la un oscilator ce-i furnizează semnalul de ceas și la sursa de alimentare de +5 V, în sistem mai apar următoarele circuite: o memorie RAM de 256 × 8 biți ce este necesară pentru memorarea variabilelor și pentru a asigura lucrul cu stiva, o memorie PROM de 2K × 8 în care se află programul de lucru și un circuit de I/E specializat Z80—PIO (Parallel Input-Output) cu ajutorul căruia sistemul poate culege și, respectiv, transmite informații paralele pe 8 biți pe cele două *port*-uri de intrare/ieșire ale circuitului.

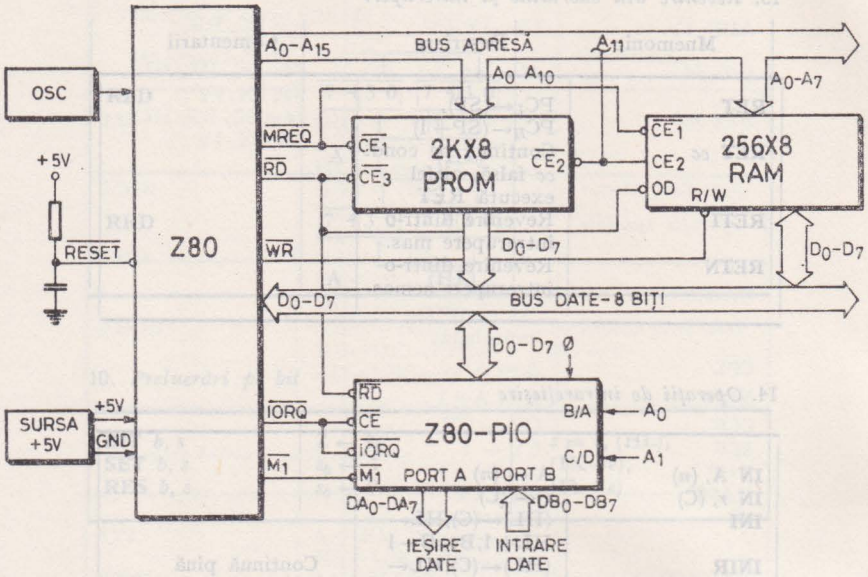


Fig. 3.8. Configurația minimă a unui microcalculator Z80

BIBLIOGRAFIE

1. * * * *The 32 bit mini*, Systems International, September 1981.
2. * * * *Micro peripherals*, Systems International, October 1981, p. 39—42.
3. * * * *Survey of 16 Bit micros-Micro movements*, Systems International, October 1981, p. 45—47.
4. * * * *Intel to extend μP family to 32 bits*, Electronic Design, June 21, 1980, p. 41, 42.
5. * * * *Microprocessor Survey-Eighties standards*, Systems International, May 1979, p. 43—45.
6. BURSKY, D., *Microprocessor Data Manual*, Electronic Design 24, November 22, 1979, p. 49—113.
7. * * * *Microcomputer components-The 5MHz 8085A-2 is the newest, highest performance 8-bit microprocessor currently available from any source*, Intel Preview, July/August 1978, p. 10, 11.
8. * * * *Microcomputer components-High-performance memory and I/O peripherals announced from MCS-85™ Microcomputer Systems*, Intel Preview, November/December 1978, p. 14, 15.
9. * * * Intel-*Component Data Catalog 1979*.
10. * * * Intel — *Component Data Catalog 1981*.
11. KLINGMAN, E.E., *Comparisons and trends in microprocessor architecture*, Computer Design, September 1977, p. 83—91.
12. BURSKY, D., *Intel sees software modules, 32-bit μP in its future*, Electronic Design, March 29, 1980.
13. * * * *Intel 8080 Microcomputer systems user's manual*, July 1975.
14. * * * *Intel MCS-85™ User's manual*, Intel Corporation, 1978.
15. BUSHELL, P., *Intel 8085 microprocessor — a review*, Microprocessors, vol. 1, no. 8, December 1977.
16. MORSE, S.P.; RAVANEL, B.W.; MAZOR, S. and POHLMAN, W.P., Intel Corporation, *Intel Microprocessors-8008 to 8086*, Computer, October, 1980.
17. SOUCEK, B., *Microprocessors and microcomputers*, John Wiley & Sons, Inc., 1976.
18. * * * Pro-log, *Microprocessor user's guide-1979/1980*, Pro-Log Corporation, July 1979.

19. * * * Zilog, *Z80-CPU, Z80A-CPU — Product specification*, March 1978.
20. * * * Zilog, *Z80-SIO, Z80A — SIO — Product specification*, August 1978.
21. * * * *Mostek Z80 Technical manual-MK 3881 Parallel I/O Controller*, August 1976.
22. * * * Zilog, *Z80™ CPU — Instruction set.*
23. * * * *Mostek Z80 Technical manual — MK 3880 Central Processing Unit*, 1977.
24. * * * *Mostek Z80 Microcomputer products*, Mostek Corporation, 1976.
25. * * * Motorola, *M6800 Aplication manual*, Motorola Inc., 1975.
26. COLLINS, D.C.; GAREN, E.R.; LAZAR, L., *Motorola's 6800 vs Intel's 8080 a side — by — side comparison*, September 1977.
27. MASATOSHI, S.; FEDERICO, F. and STANLEY, M., *An N-Channel 8-Bit Single Chip Microprocessor* 8080, 1974 IEEE International Solid-State Circuits Conference Session VI:LSI Logic.
28. DANCEA, I., *Microprocesoare — Arhitectură internă, programare, aplicații*, Editura Dacia, Cluj Napoca, 1979.
29. MC. CRACKEN, D.D., *A Guide to Intellec Microcomputer Development Systems*, Intel Corporation, 1978.
30. * * * *Microprocessor development systems' survey*, Systems International, November 1980, p. 43—45.
31. * * * *Microcomputer components-The MCS-86™ 16-bit microprocessor family is designed to deliver 10-times the performance of the industry-standard 8080*, Intel Preview, September October 1978, p. 1÷4.
32. * * * *Microcomputer systems-Total software support is available for the MCS-86 16-bit microcomputer family*, Intel Preview, March/April 1979, p. 8÷10.
33. * * * *Microcomputer components-The new, 8088 microprocessor is a powerful and flexible 8-bit CPU with full 16-bit internal architecture. Extensive bit, byte, word and string operations make it a high-performance processor*, Intel Preview, March/April 1979, p. 12, 13.
34. GLADSTONE, B.E., *Comparing microcomputer development system capabilities*, Computer Design, February 1979, p. 83—90.
35. * * * *16-bit microprocessor uses 32-bit internal processing capability for upward expansion*, Computer Design, November 1979, p. 204.
36. GUPTA, B.K., *Arithmetic processor chips enhance microprocessor system performance*, Computer Design, July 1980, p. 85—94.
37. ȚEPELEA, V.F.; ROMAN, D.N., *Aspecte privind proiectarea cu circuite integrate pe scară largă-microprocesoare*, comunicare la sesiunea Electronica în cincinalul revoluției tehnico-științifice în România, I.C.P.E., București, 4—6 Noiembrie 1976.

SD-8080, UN SISTEM DE DEZVOLTARE PENTRU MICROPROCESOARE

4.1. ROLUL SISTEMULUI DE DEZVOLTARE

Un sistem de dezvoltare este un microcalculator martor realizat cu un anumit tip de microprocesor; posedînd bogate resurse hardware și software, el folosește proiectanților în realizarea de sisteme dedicate pentru o anumită aplicație. Microsistemele specializate vor avea, de regulă, o amploare mai restrînsă decît sistemul de dezvoltare.

Menționăm că pornind de la convingerea utilității unui asemenea sistem am realizat sistemul de dezvoltare denumit SD-8080 [26]. El este construit în jurul unui microprocesor Intel 8080A și poate fi folosit atît pentru realizarea proiectelor bazate pe acest tip de microprocesor, cît și ca sistem suport pentru proiectele bazate pe familia de microprocesoare bipolare Am 2900.

În calitatea sa de sistem de dezvoltare, SD-8080 se caracterizează din punct de vedere hardware prin:

a. memorie RAM suficient de mare pentru punerea la punct a programelor de aplicație;

b. memorie PROM conținînd monitorul și interpretorul BASIC;

c. posibilitate de a lucra în întreruperi, un rol special fiind atribuit întreruperii-panou;

d. posibilitate de lucru în acces direct la memorie (DMA);

e. panou de comandă pentru vizualizarea stării microprocesorului și magistralelor;

f. gamă largă de periferice, inclusiv memorie externă pe casete magnetice;

g. un acces la dispoziția utilizatorului permițînd conectarea la SD-8080 a oricărui hardware specializat.

Din punct de vedere software SD-8080 e caracterizat prin:

a. posibilitatea de a modifica după voie conținutul memoriei și registrelor, de a lansa în execuție programe, ca și de a examina în orice punct stadiul execuției unui program, lucruri care se realizează cu ajutorul monitorului;

b. posibilitatea de a scrie programe în limbajul de asamblare 8080, care apoi sînt transformate de programul-asamblor în cod-mașină direct executabil pe SD-8080 sau pe sistemul dedicat;

c. posibilitatea de a scrie programe în limbajul conversațional de nivel înalt BASIC.

Să vedem acum în ce mod sistemul de dezvoltare ne poate fi de folos în cursul proiectării și realizării unui sistem dedicat bazat pe microprocesor. În primul rînd, el constituie baza de plecare necesară proiectantului care trece de la proiectarea logică clasică cu porți și bistabili la microprocesoare. Această trecere e marcată în special de familiarizarea cu tehnicile de programare, domeniu în care sistemul de dezvoltare poate fi un instrument prețios.

Concentrîndu-ne acum asupra proiectării hardware a unui microcalculator dedicat (fig. 4.1) putem considera că orice asemenea microcalculator este alcătuit din două părți distincte:

a. *Nucleul microcalculatorului* constituit din microprocesor și logica ce-l servește (ceas și circuite de comandă a *bus*-urilor asociate cu un minimum de memorie). Observăm că nucleul este de fapt hardware-ul minimal pe care se poate rula un program. Nucleul respectiv va fi întîlnit în toate microcalculatoarele dedicate, el fiind același, indiferent de tipul aplicației tratate. Din aceste considerente, partea microcalculatorului la care ne referim va putea constitui un proiect tip.

b. *Hardware-ul specializat* formează partea care se adaugă la nucleu pentru a realiza un microcalculator dedicat, adică partea mașinii care va fi specifică pentru aplicația dată. Conectarea hardware-ului specializat la nucleu se face pe o interfață care este de asemenea tipizată fiind denumită, în cazul lui SD-8080, *acces la dispoziția utilizatorului.*

Pentru a concretiza aceste noțiuni să luăm ca exemplu un sistem bazat pe microprocesor pentru reglarea temperaturii într-o incintă. El va fi compus din microprocesor, memorie, un bloc de conversie analog-numerică pentru introducerea valorilor de temperatură în microcalculator și un bloc de reglare a curentului prin rezistențele de încălzire (fig. 4.1). Va mai exista, de asemenea, un display, pentru a permite intervenția operatorului și pentru semnalarea condițiilor de funcționare normală sau de avarie (de exemplu, depășirea limitelor extreme de temperatură admise).

În ceea ce privește microprocesorul și circuitele ce-l deservesc nu există nici o problemă de proiectare, se reproduce proiectul tip. Se proiectează după aceea hardware-ul specializat: convertorul analog-numeric, multiplexorul, circuitele de legătură cu termorezistențele, circuitele de comandă a curentului prin rezistențele de încălzire, în așa fel încît toate aceste componente să se conecteze la nucleu pe interfața tipizată (fig. 4.2).

Sistemul de dezvoltare va fi de mai mare folos în proiectarea hardware prin faptul că el pune la dispoziție nucleul gata realizat, pînă la nivelul inter-

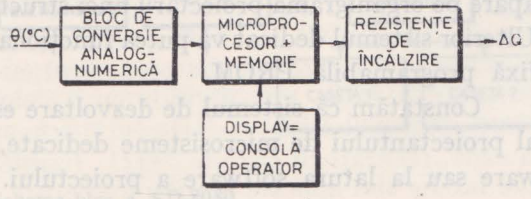


Fig. 4.1. Microsistem pentru reglarea temperaturii într-o incintă

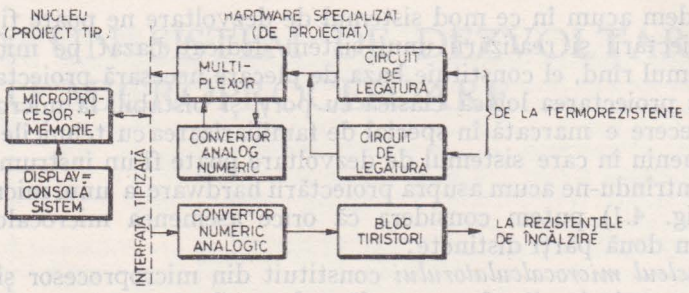


Fig. 4.2. Identificarea nucleului și a hardware-ului specializat în cazul microsistemului din figura 4.2.

feței tipizate. Proiectantul va realiza deci, în primul rînd, hardware-ul specializat, îl va conecta la sistemul de dezvoltare și-l va putea astfel experimenta și pune la punct, chiar dacă nu are de la început întreg sistemul realizat.

În ceea ce privește scrierea programelor, sistemul de dezvoltare intervine în primul rînd în faza de compilare a programelor, care se face direct pe SD-8080, fără a mai face apel la un calculator mai mare.

În faza de simulare a programelor, sistemul de dezvoltare ușurează mult sarcina proiectantului; el reprezintă instrumentul ideal pentru simulare, deoarece execută în timp real și fără nici o interpretare setul de instrucțiuni al microprocesorului. Simularea este completă, pentru că se poate testa nu numai corectitudinea semantică a programelor, ci și timpul de execuție, care poate fi deseori un element critic. Reluînd exemplul precedent, al unui sistem bazat pe microprocesor de reglare a temperaturii într-o incintă, să considerăm că datele achiziționate de sistemul de conversie analog-numerică trebuie convertite în virgulă mobilă și liniarizate, apoi, pe baza datelor de intrare, microsistemul va trebui să calculeze noua valoare a curentului de încălzire și eventual să alarmeze operatorul sau să răspundă la vreo solicitare de dialog cu consola; toate aceste operații trebuie să se încadreze într-o cuantă de timp impusă. Testarea pe sistemul de dezvoltare are în acest caz mare importanță pentru că este singurul mod de a măsura timpul de execuție al programelor de aplicație în diferitele cazuri reale ce apar în exploatare.

Validarea corectitudinii întregului proiect se va face cu ajutorul sistemului de dezvoltare la care s-a cuplat hardware-ul specializat, rulînd programele care implementează aplicația dată, aducînd astfel la îndeplinire și ultima fază care apare pe organigrama proiectării unei structuri cu microprocesor (vezi cap. 2). Ulterior sistemul dedicat va putea funcționa cu programele înscrise în memoria fixă programabilă PROM.

Constatăm că sistemul de dezvoltare este un auxiliar deosebit de prețios al proiectantului de microsisteme dedicate, fie că ne referim la latura hardware sau la latura software a proiectului.

4.2. RESURSE HARDWARE

4.2.1. PREZENTARE GENERALĂ

Așa cum s-a arătat, sistemul de dezvoltare SD-8080 este construit în jurul unui microprocesor Intel 8080A. În figura 4.3, unde se prezintă schema-bloc a SD-8080, microprocesorul este deservit de circuitul de ceas care-i furnizează semnalele de ceas Ø1 și Ø2 cu frecvența de 2 MHz, de logica de stare și comandă a magistralelor și de circuitul de tratare a întreruperilor; toate acestea pot fi asimilate cu unitatea centrală a unui calculator clasic. Celelalte elemente ale sistemului de dezvoltare sînt conectate la unitatea centrală prin intermediul celor trei magistrale: de date, formată din 8 linii bidirecționale, de adrese, care are 16 linii unidirecționale, și de comandă ce are 5 linii și e rezultatul decodificării octetului de stare trimis de microprocesor la începutul fiecărui ciclu-mașină.

Memoria aferentă acestei unități centrale are capacitatea maximă admisibilă pentru microprocesorul 8080, adică 64 Kocteți, dintre care 20 Kocteți memorie fixă PROM și restul de 44 Kocteți memorie RAM; de menționat că în PROM sînt rezidente monitorul și interpretorul BASIC.

Remarcăm prezența *accesului la dispoziția utilizatorului*, a cărui necesitate în cadrul sistemului de dezvoltare a fost comentată în paragraful anterior; el este constituit în principal din trei magistrale la care se mai adaugă pentru comoditatea proiectantului și alte semnale utile din sistem.

Dacă luăm în considerare și interfața de I/E pentru consolă, indispensabilă funcționării monitorului, am obținut *nucleul sistemului de dezvoltare* în accepțiunea pe care am dat-o acestei noțiuni în paragraful 4.1.

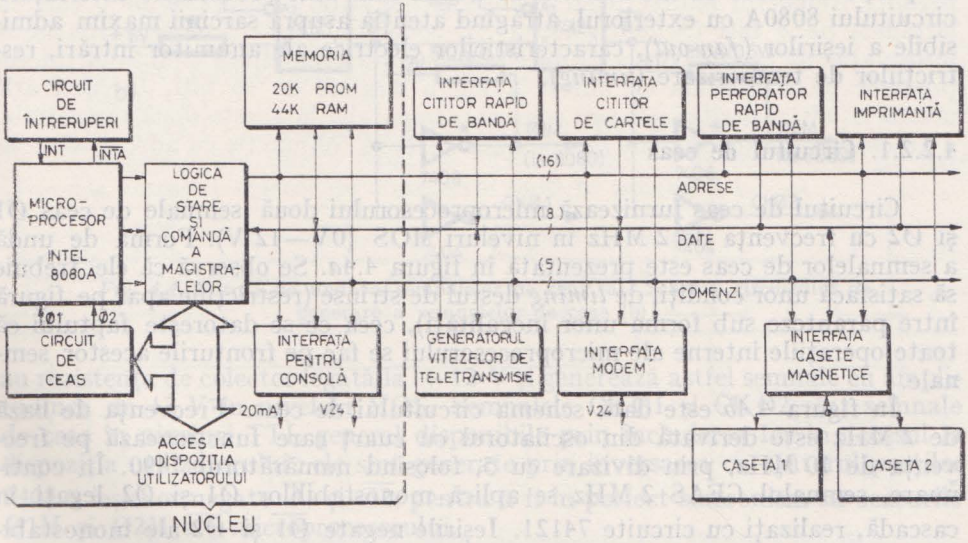


Fig. 4.3. Schema-bloc a SD-8080

Pe schema-bloc din figura 4.3 mai apar și alte interfețe de I/E cum ar fi: interfața pentru modem, interfața pentru cititorul rapid de bandă, interfața pentru perforatorul rapid de bandă, interfața pentru cititorul de cartele, interfața pentru imprimantă, interfața pentru cele două unități de casete magnetice. Acestea completează posibilitățile oferite de SD-8080 pentru crearea, compilarea și punerea la punct a programelor de aplicație scrise în limbaj de asamblare sau în BASIC.

4.2.2. MICROPROCESORUL ȘI CIRCUITELE AFERENTE

Așa cum am văzut în capitolul 3, microprocesorul 8080A este un circuit integrat MOS, pe scară largă, cu 40 de pini. Pentru ca acest circuit să poată îndeplini funcțiunea de unitate centrală pentru care este proiectat, el trebuie mai întîi alimentat cu trei tensiuni continue: +12 V, +5 V și −5 V; apoi este necesar să i se asigure semnalele de ceas Ø1 și Ø2 în niveluri MOS și, în sfîrșit, să se organizeze fluxul corespunzător al semnalelor de date, adrese și comandă pentru interfața cu memoria și perifericele. Altfel spus, circuitul 8080A în sine reprezintă o veritabilă unitate centrală pe 8 biți numai dacă este înconjurat de un număr de circuite externe care îl deservesc și pe care le vom descrie în continuare.

Atragem atenția că aceste funcții externe pot fi în cea mai mare parte realizate cu circuitele specializate Intel 8224 — Clock Generator și Intel 8228/8238 — System Controller and Bus Driver. La proiectarea lui SD-8080 circuitele respective nu erau disponibile și de aceea implementarea acestor funcții a fost făcută folosind o variantă mai puțin economică. Am considerat totuși utilă introducerea în text a prezentării variantei realizate convențional cu porți, deoarece ea are meritul de a fixa mai bine problemele interfațării circuitului 8080A cu exteriorul, atrăgînd atenția asupra sarcinii maxim admisibile a ieșirilor (*fan-out*), caracteristicilor electrice ale anumitor intrări, restricțiilor de temporizare (*timing*).

4.2.2.1. Circuitul de ceas

Circuitul de ceas furnizează microprocesorului două semnale de ceas Ø1 și Ø2 cu frecvența de 2 MHz în niveluri MOS (0V—12 V). Forma de undă a semnalelor de ceas este prezentată în figura 4.4*a*. Se observă că ele trebuie să satisfacă unor condiții de *timing* destul de strînse (restricțiile apar pe figură între paranteze sub forma unor inegalități), ceea ce se datorește faptului că toate operațiile interne ale microprocesorului se fac pe fronturile acestor semnale.

În figura 4.4*b* este dată schema circuitului de ceas. Frecvența de bază de 2 MHz este derivată din oscilatorul cu cuarț care funcționează pe frecvența de 10 MHz, prin divizare cu 5, folosind numărătorul 7490. În continuare, semnalul CEAS 2 MHz se aplică monostabililor Ø1 și Ø2 legați în cascadă, realizați cu circuite 74121. Ieșirile negate $\overline{Ø1}$ și $\overline{Ø2}$ ale monostabililor fazelor sînt aplicate porților inversoare cu colector în gol de tip 7406 care

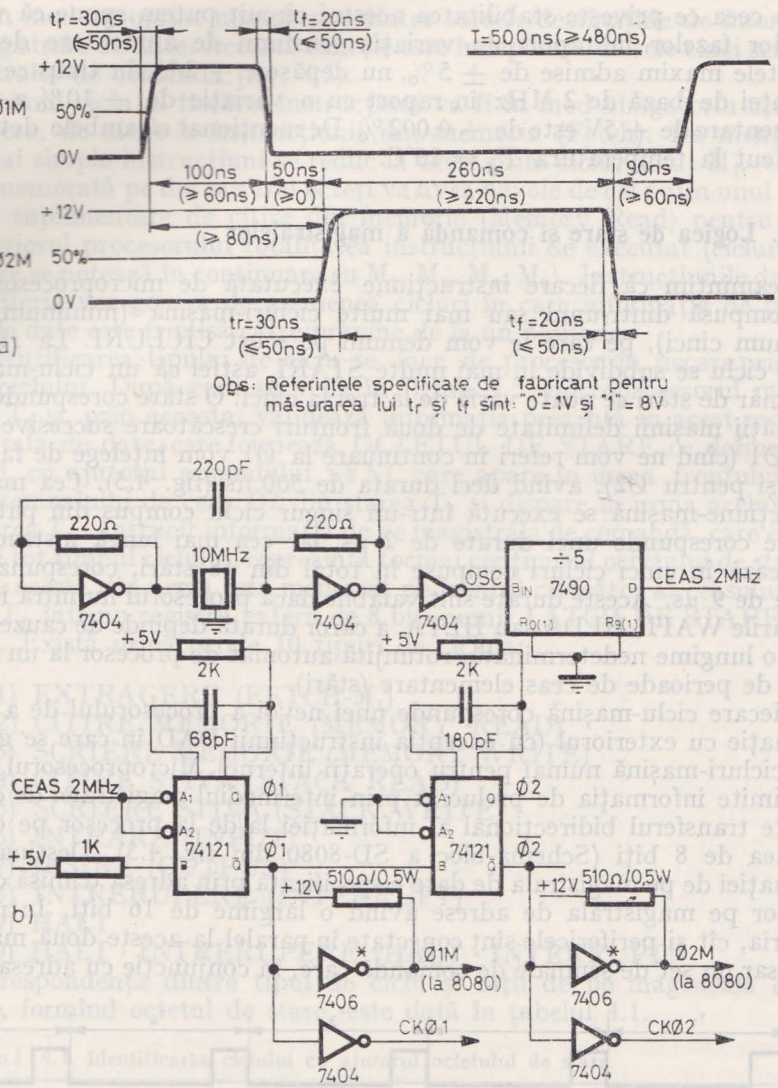


Fig. 4.4. Forma de undă a semnalelor de ceas (a). Schema circuitului de generare a semnalelor de ceas (b)

au rezistența de colector legată la + 12 V și generează astfel semnale cu amplitudinea de 12 V în niveluri MOS. Semnalele CKØ1 și CKØ2 sînt semnale de ceas în niveluri TTL general disponibile prin includerea lor în accesul la dispoziția utilizatorului; ele sînt generate prin inversarea cu ajutorul porților 7404 a ieșirilor negate $\overline{Ø1}$ și $\overline{Ø2}$, pentru a fi în perfect sincronism cu ceasurile Ø1M și Ø2M ale microprocesorului.

În ceea ce privește stabilitatea acestui circuit putem spune că variațiile duratelor fazelor în raport cu variația tensiunii de alimentare de + 5 V, în limitele maxim admise de ± 5%, nu depășesc ± 2%, în timp ce variația frecvenței de bază de 2 MHz în raport cu o variație de ± 10% a tensiunii de alimentare de + 5V este de + 0,002%. De menționat că ambele determinări s-au făcut la temperatura $T_A = 25°C$.

4.2.2.2. Logica de stare și comandă a magistralelor

Reamintim că fiecare instrucțiune executată de microprocesorul 8080 este compusă dintr-unul sau mai multe cicluri-mașină (minimum unul și maximum cinci), pe care le vom denumi pe scurt CICLURI. La rîndul lui, fiecare ciclu se subdivide în mai multe STĂRI, astfel că un ciclu-mașină are un număr de stări ce poate varia de la trei la cinci. O stare corespunde în timp activității mașinii delimitate de două fronturi crescătoare succesive ale ceasului Ø1 (cînd ne vom referi în continuare la Ø1 vom înțelege de fapt Ø1M, la fel și pentru Ø2), avînd deci durata de 500 ns (fig. 4.5). Cea mai scurtă instrucțiune-mașină se execută într-un singur ciclu compus din patru stări, ceea ce corespunde unei durate de 2 μs, iar cea mai lungă instrucțiune se efectuează în cinci cicluri compuse în total din 18 stări, corespunzînd unei durate de 9 μs. Aceste durate sînt valabile dacă procesorul nu intră în vreuna din stările WAIT, HLDA sau HLTA, a căror durată depinde de cauze externe, avînd o lungime nedeterminată, rotunjită automat de procesor la un multiplu întreg de perioade de ceas elementare (stări).

Fiecare ciclu-mașină corespunde unei nevoi a procesorului de a schimba informație cu exteriorul (cu excepția instrucțiunii DAD în care se generează două cicluri-mașină numai pentru operații interne). Microprocesorul primește sau trimite informația de prelucrat prin intermediul magistralei de date care permite transferul bidirecțional al informației la/de la procesor pe o cale cu lărgimea de 8 biți (Schema-bloc a SD-8080 din fig. 4.3). Destinația/sursa informației de pe magistrala de date e specificată prin adresa trimisă de microprocesor pe magistrala de adrese avînd o lărgime de 16 biți. Întrucît atît memoria, cît și perifericele sînt conectate în paralel la aceste două magistrale, e necesar un set de semnale de comandă care, în conjuncție cu adresa, să acti-

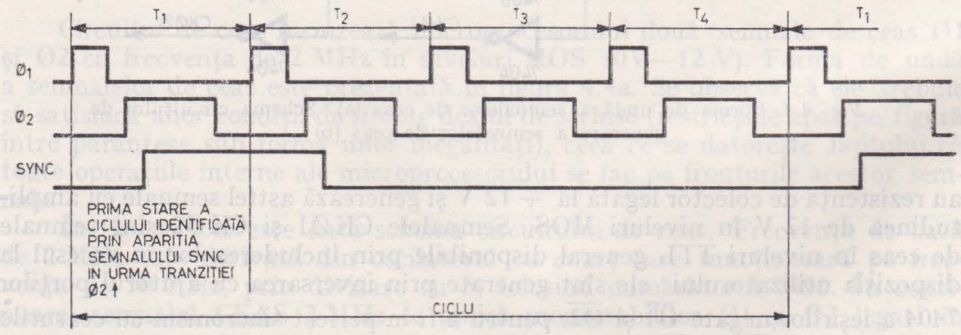


Fig. 4.5. Structura unui ciclu-mașină format din patru stări: T_1, T_2, T_3 și T_4

veze în mod apropriat numai una dintre sursele/destinațiile de informație de pe magistrala de date. Această funcție este îndeplinită de cele cinci semnale ale magistralei de comandă (fig. 4.3).

Primul ciclu al fiecărei instrucțiuni va fi în mod obligatoriu un ciclu de extragere, de citire a instrucțiunii din memorie (Fetch). De menționat că cele mai simple instrucțiuni se reduc la acest prim ciclu, notat M_1. O instrucțiune memorată pe doi sau trei octeți va avea nevoie de cel puțin unul sau două cicluri suplimentare de citire din memorie (Memory Read) pentru a aduce în interiorul procesorului totalitatea instrucțiunii de executat (ciclurile suplimentare se notează în continuare cu M_2, M_3, M_4, M_5). Instrucțiunile de transfer cu periferia vor genera de asemenea cicluri în care informația de pe magistrala de date este trimisă către/provine de la un periferic.

Identificarea tipului de ciclu se face de procesor în fiecare primă stare, T_1, a ciclului. După cum se vede în figura 4.5, microprocesorul marchează stările T_1 și, prin aceasta, validează informația prezentă în acest moment pe magistrala de date, care formează OCTETUL DE STARE ce definește tipul ciclului, cu ajutorul semnalului SYNC care apare în urma frontului crescător al semnalului de ceas Ø2 din starea T_1 și dispare în urma aceluiași front din starea T_2. Întrucît informația de pe magistrala de comandă, care este necesară pe tot timpul ciclului, reprezintă tocmai conținutul octetului de stare decodificat, acesta din urmă este memorat pe frontul crescător al ceasului Ø1 din starea T_2 într-un registru extern de 8 biți numit LATCH DE STARE (Status Latch). Există următoarele 10 tipuri de cicluri:

(1) EXTRAGERE (FETCH-M_1)
(2) CITIRE MEMORIE (MEMORY READ)
(3) SCRIERE MEMORIE (MEMORY WRITE)
(4) CITIRE STIVĂ (STACK READ)
(5) SCRIERE STIVĂ (STACK WRITE)
(6) INTRARE (INPUT)
(7) IEȘIRE (OUTPUT)
(8) ÎNTRERUPERE (INTERRUPT)
(9) HALT
(10) HALT · ÎNTRERUPERE (HALT · INTERRUPT)

Corespondența dintre tipul de ciclu și biții de pe magistrala de date, D_0—D_7, formînd octetul de stare, este dată în tabelul 4.1.

Tabelul 4.1. **Identificarea ciclului cu ajutorul octetului de stare**

Bitul	Semnificația în octetul de stare	Tipul de ciclu									
		1	2	3	4	5	6	7	8	9	10
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

Pentru o mai bună înțelegere a mecanismului de identificare a tipului de ciclu vom descrie în continuare semnificația fiecărui bit component al octetului de stare, comentînd motivele prezenței sau absenței lui în cazul diferitelor tipuri de cicluri.

INTA (INTERRUPT ACKNOWLEDGE) este bitul de acceptare a unei cereri de întrerupere (semnalul INT) prezentată procesorului în timp ce sistemul de întreruperi este activat (bistabilul intern INTE = „1"). Acest bit apare în timpul ciclurilor ÎNTRERUPERE sau HALT · ÎNTRERUPERE semnificînd faptul că întreruperea a fost luată în considerare în timpul unei funcționări normale a programului sau, respectiv, în timp ce microprocesorul era blocat în starea HALT (HLTA = „1"). În cursul unui astfel de ciclu, procesorul va executa instrucțiunea forțată din exterior pe magistrala de date, instrucțiune de tip RESTART sau CALL, care va face ca procesorul să înceapă execuția subrutinei de întrerupere aferente cauzei care a produs-o. Acesta este motivul pentru care bitul M_1, ce indică un ciclu de extragere, primul ciclu al instrucțiunii, este prezent în ambele cazuri.

$\overline{WO}$ (WRITE OUTPUT) evidențiază faptul că ciclul care debutează va fi un ciclu de transfer al informației dinspre procesor spre exterior (sensurile de transfer al informației sînt referențiate în raport cu procesorul). Este un semnal activ jos ce apare în ciclurile de tip SCRIERE MEMORIE, SCRIERE STIVĂ (stiva fiind plasată tot în memorie, din exteriorul procesorului aceste două tipuri de ciclu nu se deosebesc prin nimic) și IEȘIRE în care procesorul va scrie informație în memorie sau pe un dispozitiv periferic. După cum vom vedea mai tîrziu, informația apare pe magistrala de date în timpul stării T_3 a ciclului respectiv, strobată cu semnalul $\overline{WR}$ (Write) generat de procesor.

STACK este bitul care diferențiază ciclurile CITIRE STIVĂ și SCRIERE STIVĂ de ciclurile CITIRE MEMORIE și respectiv SCRIERE MEMORIE. Întrucît stiva este plasată în memoria principală, iar evidența lucrului cu stiva este ținută de către procesor, privind din exteriorul procesorului aceste două cicluri sînt două cîte două echivalente. Bitul STACK, care nu se reflectă mai departe pe magistrala de comandă, este vizualizat și poate fi folosit în scop de test.

HLTA (HALT ACKNOWLEDGE) este prezent în ciclurile de tip HALT sau HALT · ÎNTRERUPERE, semnalînd exteriorului că procesorul s-a blocat în urma executării unei instrucțiuni HALT. Deblocarea procesorului poate avea loc fie printr-o întrerupere externă, dacă ea poate fi acceptată de procesor, fie ca urmare a unei inițializări generale a procesorului produsă de semnalul RESET.

OUT (OUTPUT) indică faptul că se execută o operație de ieșire pe un periferic și apare numai în ciclurile de tip IEȘIRE. Informația de pe magistrala de date e validată de procesor în starea T_3 cu ajutorul semnalului $\overline{WR}$ (activ jos).

M_1 este bitul care marchează totdeauna primul ciclu al unei instrucțiuni (așa cum SYNC marchează prima stare a unui ciclu); el apare în mod natural în ciclurile de EXTRAGERE și, de asemenea, așa cum am mai explicat, în ciclurile ÎNTRERUPERE și HALT · ÎNTRERUPERE în care informația

forțată din exterior pe magistrala de date este interpretată ca primul octet al unei instrucțiuni.

INP (INPUT) indică faptul că se execută o operație de intrare de la un periferic și apare numai în ciclurile de tip INTRARE. Informația de intrare va fi prezentată de către dispozitivul de intrare pe magistrala de date, la activarea de către procesor a semnalului DBIN (Data Bus In).

MEMR (MEMORY READ) identifică un ciclu de citire a informației de la memorie. La fel ca și în cazul ciclului de INTRARE informația de la memorie va fi prezentă pe magistrala de date pe timpul lui DBIN. Pentru a înțelege mai bine funcționarea lui MEMR vom comenta cazurile cînd el *nu* este prezent și anume în ciclurile de SCRIERE (MEMORIE sau STIVĂ), în ciclurile de INTRARE sau IEȘIRE în care se lucrează cu un dispozitiv periferic și în cazul ciclurilor ÎNTRERUPERE și HALT · ÎNTRERUPERE în care primul octet, codul instrucțiunii, nu provine de la memorie, ci este forțat printr-o logică specializată pe magistrala de date, pentru a putea indica în acest fel procesorului cauza care a produs întreruperea.

În figura 4.6 este prezentată cronograma unui ciclu-mașină. Așa cum s-a mai văzut, starea T_1, în care apare semnalul SYNC, este dedicată identificării tipului de ciclu ce debutează. Pe magistrala de date D_7—D_0 procesorul trimite octetul de stare care poate fi preluat pe frontul crescător al lui Ø1 din T_2, memorat și decodificat pentru a da naștere semnalelor magistralei de comandă. Pe magistrala de adrese A_{15}—A_0 procesorul trimite fie adresa de memorie a locației care trebuie citite sau scrise, fie adresa perifericului cu care se va lucra (întrucît adresa de I/E are numai 8 biți ea va fi repetată pe ambele jumătăți ale magistralei: A_{15}—A_8 și A_7—A_0). Această informație este stabilă pe magistrala de adrese în stările T_2, eventual T_W și T_3. Starea T_2 servește în principal ca timp de efectuare a accesului la memorie/periferic în cazul ciclurilor care presupun citirea de la o sursă de informație externă procesorului. După căderea semnalului SYNC la apariția lui Ø2 din T_2, procesorul eliberează magistrala de date pe care o folosise pentru trimiterea octetului de stare și care acum va fi folosită pentru alte transferuri de informație. Considerînd cazul unui ciclu de citire, după căderea lui SYNC și eliberarea magistralei de date, procesorul ridică semnalul DBIN și inversează sensul magistralei pentru a permite intrarea datelor. Pe frontul căzător al ceasului Ø2 el inspectează starea liniei de intrare READY care este folosită de sursa externă de informații pentru sincronizarea dispozitivelor cu timp de acces mai mare ca durata stării T_2. Dacă sursa externă de informații nu poate furniza datele în timpul cerut, ea va face linia READY = „0", ceea ce va cauza intrarea procesorului într-o stare temporară de așteptare, WAIT, notată pe figură cu T_W. Părăsirea acestei stări are loc de îndată ce se întîlnește READY = „1" pe frontul căzător al lui Ø2 din T_W. Condiția care dictează numărul de stări T_W, cu alte cuvinte durata pentru care READY = „0" după inițierea accesului în T_2, este ca datele citite să fie stabile pe magistrala *procesorului* cu un timp de stabilire (*setup*) determinat înainte de frontul crescător al ceasului Ø1 din T_3, cînd conținutul magistralei de date este considerat valid și introdus în procesor. Starea T_W dispare cu desăvîrșire dacă datele pot fi pregătite de sursa externă de informații pînă la sfîrșitul lui T_2.

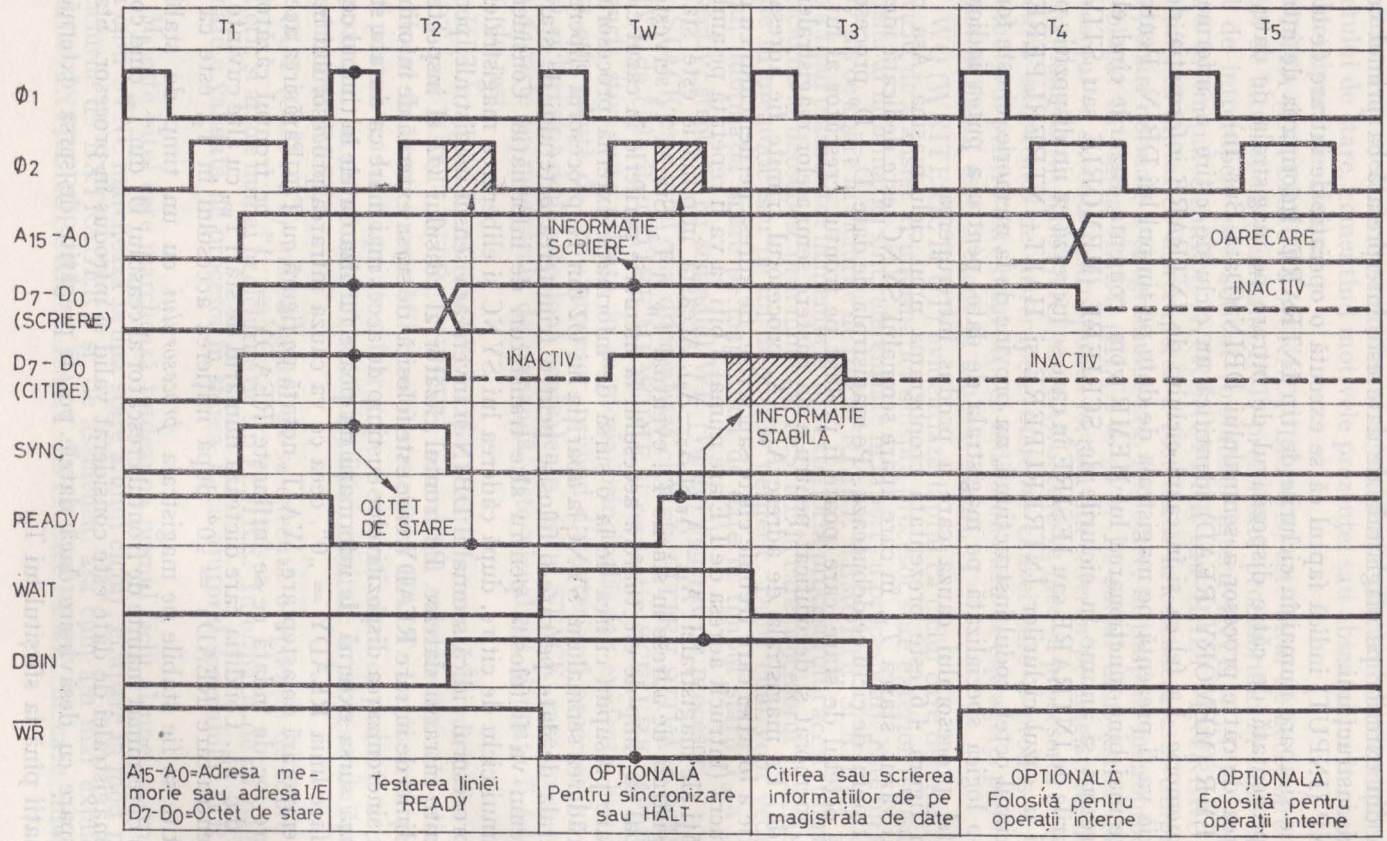


Fig. 4.6. Cronograma generală a unui ciclu-mașină

În cazul unui ciclu de scriere de informație dinspre procesor spre exterior în T_2 procesorul nu mai ridică DBIN, sensul magistralei de date rămîne neschimbat; după înlăturarea de pe magistrală a octetului de stare procesorul trimite imediat pe magistrală informația pe care dorește să o scrie și apoi testează READY, la fel ca și în cazul unui ciclu de citire. Dacă destinația informației (memoria sau un dispozitiv de I/E) nu este pregătită de a accepta datele, ea va forța READY la „0", ceea ce va produce apariția stării WAIT. Notăm că în acest caz semnalul $\overline{WR}$ este activat (devine „0") încă de la apariția stării T_W. Adăugăm că starea T_W apare pentru un timp nedefinit în cazul unui ciclu de tip HALT, ea putînd fi părăsită numai printr-o întrerupere sau prin inițializare generală RESET.

Într-un ciclu de citire de la memorie/periferie, în starea T_3, procesorul preia informația de pe magistrala de date atît pe frontul crescător al ceasului Ø1, cît și pe frontul crescător al ceasului Ø2, informația trebuind să rămînă stabilă un timp determinat după apariția lui Ø2 din T_3 (*hold*). În acest caz, după căderea lui DBIN magistrala de date e trecută de procesor în starea inactivă (magistrala de date bucurîndu-se de proprietatea că circuitele de ieșire ale microprocesorului pe liniile acesteia au trei stări posibile: „0" logic, „1" logic și starea inactivă sau starea a treia în care circuitul de ieșire prezintă pentru exterior o impedanță ridicată, ceea ce echivalează practic cu excluderea lui din circuit). După căderea lui DBIN, sursa externă de informație va putea înlătura informația de pe magistrală, trecîndu-și circuitele de interfață cu aceasta în starea inactivă, astfel că magistrala va rămîne inactivă pe tot restul ciclului.

Dacă ciclul curent este de scriere în memorie sau pe un dispozitiv periferic, atunci în starea T_3 se trimite semnalul de validare a magistralei de date, $\overline{WR}$ = = „0", care comandă preluarea datelor de către destinația acestora. Informația de ieșire este înlăturată de pe magistrala de date la un moment ulterior din starea T_4.

Stările T_4 și T_5 sînt ambele opționale și servesc pentru realizarea unor operații interne ale procesorului care nu se manifestă prin nici o activitate la interfața cu exteriorul.

Semnalele magistralei de comandă sînt:

1. $\overline{MEMR}$ (Memory Read) este un semnal activ jos, generat de un circuit „3-stări", ce apare în ciclurile în care se efectuează o citire de la memorie, condiționat în timp de prezența lui DBIN. Ecuația lui este:

$$\overline{MEMR} = \overline{MEMRA \cdot DBIN},$$

unde MEMRA (corespunzînd bitului 7 din octetul de stare) selectează acele cicluri în care se efectuează o citire de la memorie.

2. $\overline{MEMW}$ (Memory Write) este un semnal activ jos, generat de un circuit de ieșire „3-stări", apărînd în ciclurile de scriere la memorie, condiționat în timp de prezența lui $\overline{WR}$. Ecuația lui este:

$$\overline{MEMW} = \overline{\overline{OUT} \cdot WR},$$

unde WR apare în toate ciclurile de ieșire în timpul lui T_3, iar $\overline{OUT}$ (negarea bitului 4 din octetul de stare) selectează acele cicluri unde nu se lucrează cu un dispozitiv periferic, rezultînd implicit că se dorește lucrul cu memoria.

3. $\overline{I/OR}$ (Input/Output Read) este un semnal activ jos cu ieșirea pe un circuit cu „3-stări" ce apare în ciclurile de citire de la un dispozitiv periferic, fiind condiționat în timp de prezența lui DBIN. Ecuația lui este:

$$\overline{I/OR} = \overline{INP \cdot DBIN},$$

unde INP (corespunzînd bitului 6 din registrul de stare) selectează ciclurile de introducere de informație de la un dispozitiv periferic.

4. $\overline{I/OW}$ (Input/Output Write) este un semnal activ jos cu ieșirea pe un circuit „3-stări" ce apare în ciclurile de scriere pe un dispozitiv periferic, fiind condiționat în timp de prezența lui $\overline{WR}$. Ecuația lui este:

$$\overline{I/OW} = \overline{OUT \cdot WR},$$

unde WR apare în toate ciclurile de ieșire în timpul lui T_3, iar OUT (corespunzînd bitului 4 din octetul de stare) selectează ciclurile în care se face o ieșire pe un dispozitiv periferic,

5. $\overline{INTA}$ (Interrupt Acknowledge) este un semnal activ jos cu ieșirea pe un circuit cu „colector în gol" care apare în ciclurile de acceptare a unei întreruperi în scopul forțării din exterior pe magistrala de date a codului unei instrucțiuni RESTART sau CALL de apelare a subrutinei de tratare a cauzei de întrerupere în activitate. Este condiționat în timp de DBIN. Ecuația lui este:

$$\overline{INTA} = \overline{INTAP \cdot DBIN},$$

unde INTAP (corespunzînd bitului 0 din octetul de stare) selectează ciclurile de acceptare a unei întreruperi.

Observăm că semnalele magistralei de comandă, în contrast cu informația conținută în octetul de stare, înglobează atît informații despre tipul de ciclu, cît și relații de timp, selectînd acel interval de timp din cadrul ciclului care e destinat pentru transmiterea/recepționarea pe magistrala de date a informației către destinațiile externe sau provenind de la sursele externe. Pe de altă parte, informația conținută în octetul de stare este mult mai bogată, în sensul că din cele 10 tipuri de ciclu posibile, numai 5 sînt reprezentate explicit prin semnalele magistralei de comandă, unele dintre ele fiind suprapuse din punct de vedere al funcționării memoriei/dispozitivelor periferice (Tabelul 4.2).

Memorarea octetului de stare și formarea semnalelor magistralei de comandă sînt realizate de circuitul reprezentat în figura 4.7 *a*. Registrul de memorare a octetului de stare este implementat cu un circuit Intel 8212 (8-BIT I/O PORT) configurat în modul de ieșire (intrarea MD a circuitului 8212 conectată la „1"). În această configurație, 8212 memorează datele de pe intrările DI_1—DI_8 pe timpul cît este selectat cu ajutorul semnalului compus $\overline{DS_1} \cdot DS_2$. Cum intrările de selecție $\overline{DS_1}$ și DS_2 sînt conectate respectiv la Ø1 (TTL) și la semnalul SYNC, care este repetarea cu ajutorul unei porți neinversoare rapide a semnalului SYNCP generat de procesor, acesta din urmă

Tabelul 4.2. **Reprezentarea tipurilor de ciclu prin semnalele magistralei de comandă**

Tipul de ciclu	$\overline{MEMR}$	$\overline{MEMW}$	$\overline{I/OR}$	$\overline{I/OW}$	$\overline{INTA}$
(1) EXTRAGERE	0	1	1	1	1
(2) CITIRE MEMORIE	0	1	1	1	1
(3) SCRIERE MEMORIE	1	0	1	1	1
(4) CITIRE STIVĂ	0	1	1	1	1
(5) SCRIERE STIVĂ	1	0	1	1	1
(6) INTRARE	1	1	0	1	1
(7) IEȘIRE	1	1	1	0	1
(8) ÎNTRERUPERE	1	1	1	1	0
(9) HALT	0	1	1	1	1
(10) HALT · ÎNTRERUPERE	1	1	1	1	0

neputînd fi exploatat deoarece are *fan-out* redus ($\cong$ 1), rezultă că datele vor fi capturate pe perioada cît Ø1 = „0" în T_1, iar SYNC este ridicat, zăvorîrea lor avînd loc la apariția lui Ø1 din T_2. Ieșirile DO_1—DO_8 ale circuitului 8212 repetă continuu ieșirile registrului de memorare a octetului de stare, deci ele conțin informația de stare începînd cu momentul ridicării lui SYNC, la care se adaugă timpul de propagare al lui 8212. Intrarea de ștergere a registrului lui 8212, $\overline{CLR}$, este conectată la semnalul BUSEN, ceea ce va face ca atunci cînd BUSEN devine „0" toți biții octetului de stare să devină „0", inhibînd imediat toate semnalele magistralei de comandă. Facem aici o mică paranteză pentru a spune că BUSEN este semnalul prin intermediul căruia un dispozitiv extern conectat la magistralele lui SD—8080 poate, folosind tehnica DMA (Direct Memory Access), să inactiveze toate magistralele sistemului de dezvoltare, rămînînd astfel singurul utilizator al memoriei și interfețelor sale de I/E. Dezactivarea magistralei de comandă cînd BUSEN devine „0", are loc în felul următor: deoarece toate ieșirile lui 8212 se șterg, rezultă că și INTA = = „0", ceea ce face ca poarta 7406, ce conduce $\overline{INTA}$ pe magistrala de comandă, să treacă imediat în starea cu „colectorul în gol", fapt ce echivalează în acest caz cu dezactivarea; circuitul inversor cu ieșire „3-stări" Intel 8226, care conduce celelalte semnale de pe magistrala de comandă, primește pe intrarea de selecție $\overline{CS}$ semnalul $\overline{BUSEN}$ = „1", ceea ce produce dezactivarea circuitelor de ieșire. Nu mai insistăm asupra celorlalte circuite de pe schemă, ele reprezentînd o implementare directă a ecuațiilor semnalelor magistralei de comandă, așa cum au fost prezentate anterior.

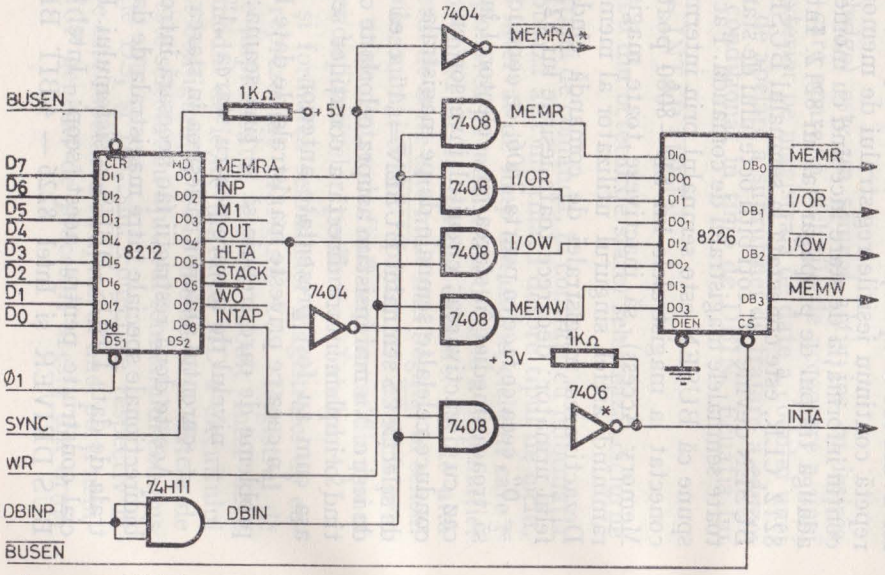
În ceea ce privește magistrala de date D_7—D_0 a procesorului, există două probleme de care trebuie să se țină seama:

— nivelul de intrare pentru starea „1" să fie de minimum 3,3 V;

— curentul absorbit la ieșirea în starea joasă „0" nu poate depăși 1,7 mA.

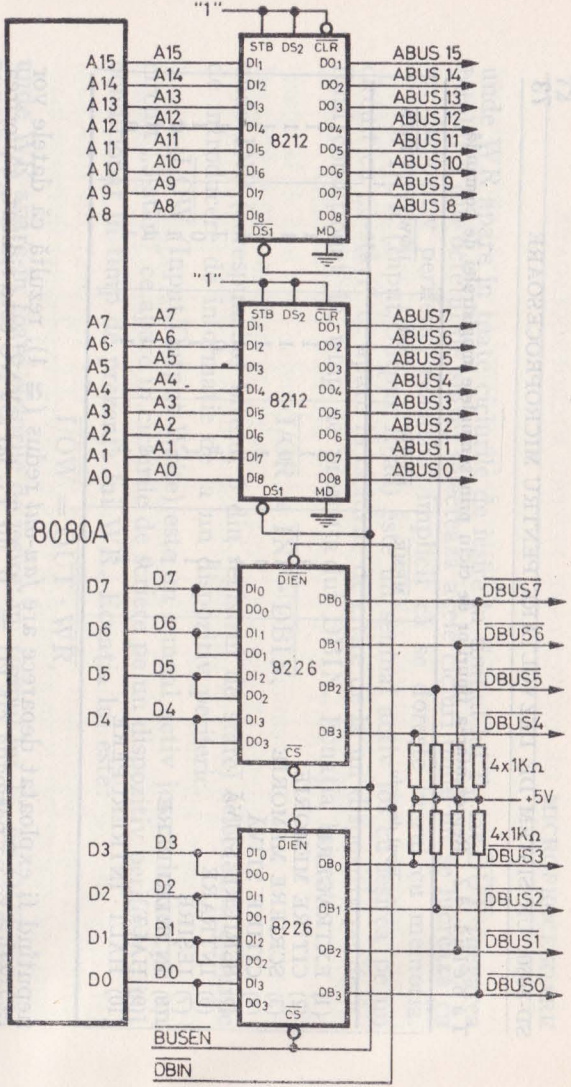
Aceste două restricții fac necesară introducerea unor circuite de comandă bidirecționale speciale între magistrala de date D_7—D_0 a procesorului și magistrala de date $\overline{DBUS_7}$-$\overline{DBUS_0}$ a sistemului. Există două tipuri de circuite special construite pentru acest scop: Intel 8216 — 4-BIT BIDIRECTIONAL BUS DRIVER și Intel 8226 — 4-BIT BIDIRECTIONAL BUS DRIVER

Fig. 4.7. Memorarea octetului de stare și formarea semnalelor magistralei de comandă (a). Circuitele de comandă ale magistralelor de comandă și de adrese (b).



a) * MEMRA este un semnal intern folosit de circuitele de selecție ale memoriei

b)



(INVERTING) care realizează specificațiile necesare funcționării magistralei procesorului 8080A, permițînd în același timp lucrul cu niveluri TTL obișnuite pe magistrala sistemului și posedînd un *fan-out* deosebit de mare (50 mA). După cum se vede în figura 4.7 *b*, în care se prezintă implementarea magistralei de date a sistemului, circuitul 8226 are două seturi de intrări/ieșiri: DI_i—DO_i (Data Input-Output) legate împreună pe magistrala procesorului, pentru a realiza dublul sens de transfer și DB_i (Data Bus) conectate pe magistrala de date a sistemului (i = 0, 1, 2, 3). Intrarea $\overline{DIEN}$ (Data Input Enable) a circuitelor 8226 stabilește sensul de transmitere al informației prin circuit după cum urmează:

— cînd $\overline{DIEN}$ = „0" (semnalul e activ jos) informația prezentă la intrarea DI_i se transmite la ieșirea DB_i (i = 0, 1, 2, 3);

— cînd $\overline{DIEN}$ = „1" informația prezentă la intrarea DB_i se transmite la ieșirea DO_i (i = 0, 1, 2, 3).

Acest semnal este conectat la DBIN, care e repetarea printr-o poartă rapidă neinversoare a semnalului DBINP furnizat de procesor, pornind de la observația că sensul de transmitere al informației este totdeauna dinspre procesor spre magistrala de date, cu excepția ciclurilor de citire cînd DBIN = = „1". Circuitele 8226 posedă și o intrare $\overline{CS}$ (Chip Select), care, atunci cînd e inactivă („1"), realizează dezactivarea circuitelor de ieșire „3-stări" de pe liniile DB, ceea ce prin conectarea acestei intrări la BUSEN permite unui dispozitiv extern să preia în totalitate controlul magistralei de date în tehnica DMA.

Precizăm că pe magistrala de date a sistemului $\overline{DBUS_7}$—$\overline{DBUS_0}$ informația este negată față de cea de pe magistrala procesorului, ceea ce nu prezintă nici un impediment pentru buna funcționare a sistemului. Mai trebuie să amintim că pe magistrala de date a sistemului toate liniile sînt dotate cu rezistențe de polarizare de 1 KΩ conectate la +5 V (reprezentînd o sarcină moderată de 5 mA în starea „0"), care servesc ca rezistențe de colector pentru circuitele de memorie PROM tip Intel 3601 ce au o ieșire de tip „colector în gol".

Partea de circuit din figura 4.7 descrisă pînă acum, adică logica de memorare a octetului de stare și de formare a semnalelor magistralei de comandă, asociată cu circuitele de comandă ale magistralei de date, poate fi realizată cu ajutorul unui singur circuit integrat specializat de tip Intel 8228/8238-SYSTEM CONTROLLER AND BUS DRIVER. Acest circuit specializat înglobează o funcțiune suplimentară referitoare la sistemul de întreruperi pe care o vom reaminti la paragraful 4.2.4.

Așa cum rezultă din figura 4.7 *b*, magistrala de adrese a SD-8080 este comandată cu ajutorul a două circuite 8212 conectate în modul de intrare (MD = „0"), astfel încît ieșirile de putere DO_i (i = 1, 2, ..., 8) urmăresc în permanență intrările DI_i (i = 1, 2, ..., 8); proprietatea de repetor logic îi este conferită de conectarea intrărilor STB (Strobe), DS_2 (Device Selection) și $\overline{CLR}$ (Clear) la „1". Magistrala de adrese a sistemului $ABUS_{15}$—$ABUS_0$ urmărește magistrala de adrese a procesorului A_{15}—A_0 atît timp cît semnalul $\overline{BUSEN}$, conectat la intrarea $\overline{DS_1}$ a circuitelor 8212, este activ („0"). Dezac-

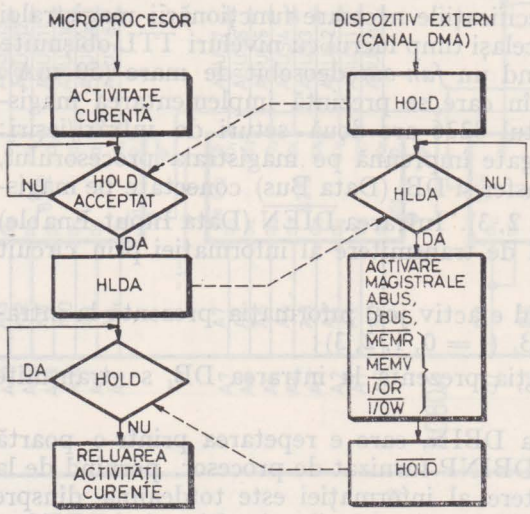


Fig. 4.8. Organigrama preluării controlului magistralelor sistemului de către un dispozitiv extern, folosind semnalele HOLD și HLDA

tivarea lui produce trecerea în starea inactivă a etajelor de ieșire „3-stări" ale circuitelor 8212, permițînd unui dispozitiv extern să preia în totalitate controlul magistralelor sistemului în tehnica DMA.

Vom detalia acum modul în care un dispozitiv extern, fie el un alt microprocesor sau un canal rapid de I/E, poate prelua controlul magistralelor sistemului SD-8080 în scopul de a folosi memoria sau dispozitivele sale periferice (fig. 4.8). Microprocesorul este prevăzut cu o linie de intrare numită HOLD, a cărei activare are ca rezultat blocarea temporară a activității procesorului, blocare ce durează atît timp cît acest semnal este activ. În momentul în care procesorul poate lua în considerare semnalul de blocare HOLD, el trimite semnalul de acceptare HLDA (Hold Acknowledge) și trece magistralele de date, adrese și comenzi ale sistemului în starea inactivă.

La recepționarea semnalului HLDA dispozitivul extern preia controlul magistralelor și execută operația pe care o dorește cu memoria sau periericele. La terminarea activității sale el va trebui să pună ieșirile circuitelor proprii de comandă a magistralelor în starea inactivă, după care readuce semnalul HOLD pe „0", ceea ce produce reluarea imediată a activității procesorului din punctul unde fusese suspendată. Ca exemplificare să considerăm canalul DMA asociat unui disc flexibil (floppy disk) căruia i se dă comanda de citire a unui sector. La inițializarea operației de citire programul de comandă trimite canalului DMA adresa unde se depun octeții citiți în memorie și numărul de octeți care se transferă, în acest caz 128. De fiecare dată ce unitatea de disc flexibil asamblează un octet canalul DMA va trece microprocesorul în starea HOLD, va efectua operația de scriere a octetului la adresa curentă, apoi va reda controlul microprocesorului. La anularea contorului de octeți canalul semnalează procesorului terminarea operației de citire-sector, de regulă printr-o întrerupere.

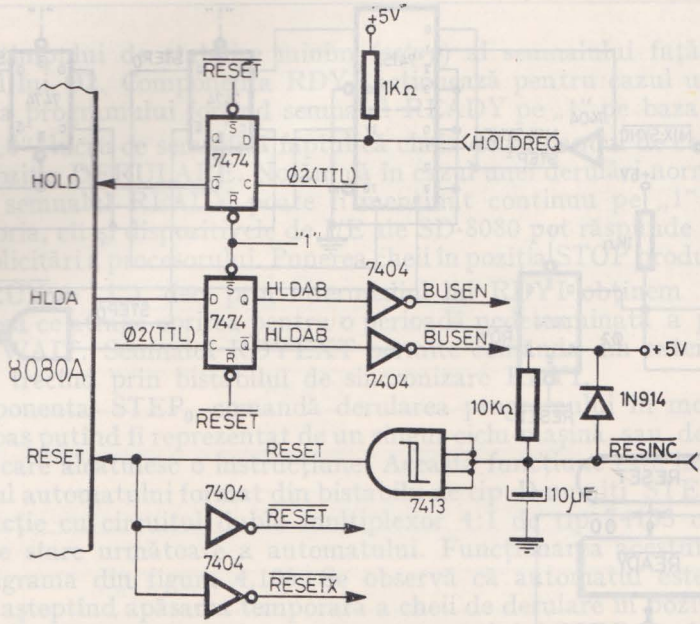


Fig. 4.9. Circuitul de tratare a semnalelor HOLD și HLDA; generarea semnalului RESET

În figura 4.9 este prezentat circuitul care tratează semnalele HOLD și HLDA; se observă că semnalul $\overline{\text{HOLDREQ}}$ (Hold Request) care este activ jos se memorează pe frontul crescător al ceasului Ø2 în bistabilul HOLD, a cărui ieșire e aplicată direct microprocesorului. În acest fel se respectă automat timpul de stabilire minim (*setup*) al semnalului HOLD față de frontul crescător al ceasului Ø2 din starea T_2 cînd procesorul testează această linie. La rîndul lui, semnalul de răspuns HLDA al microprocesorului este memorat în bistabilul HLDAB acționat pe același ceas; ieșirile acestuia sînt semnalele BUSEN și $\overline{\text{BUSEN}}$ care comandă circuitele de ieșire pe magistrale ale procesorului.

În aceeași figură este prezentat circuitul de generare a semnalului de inițializare generală a procesorului, RESET. Grupul format din rezistența de 10 KΩ, condensatorul de 10 μF și dioda de la intrare are rolul de a asigura generarea automată a semnalului de inițializare generală la aplicarea tensiunii de alimentare principale de +5 V (Power-On Reset); intrarea $\overline{\text{RESINC}}$ (activă jos) este conectată la ieșirea *latch*-ului cheii INIȚIALIZARE de pe panoul de comandă. Semnalele de ieșire ale schemei sînt RESET (activ sus) ce merge la procesor, $\overline{\text{RESET}}$ (activ jos) care e folosit în circuitele ce deservesc microprocesorul și $\overline{\text{RESETX}}$, echivalent cu $\overline{\text{RESET}}$, dar generat separat, care merge în restul sistemului.

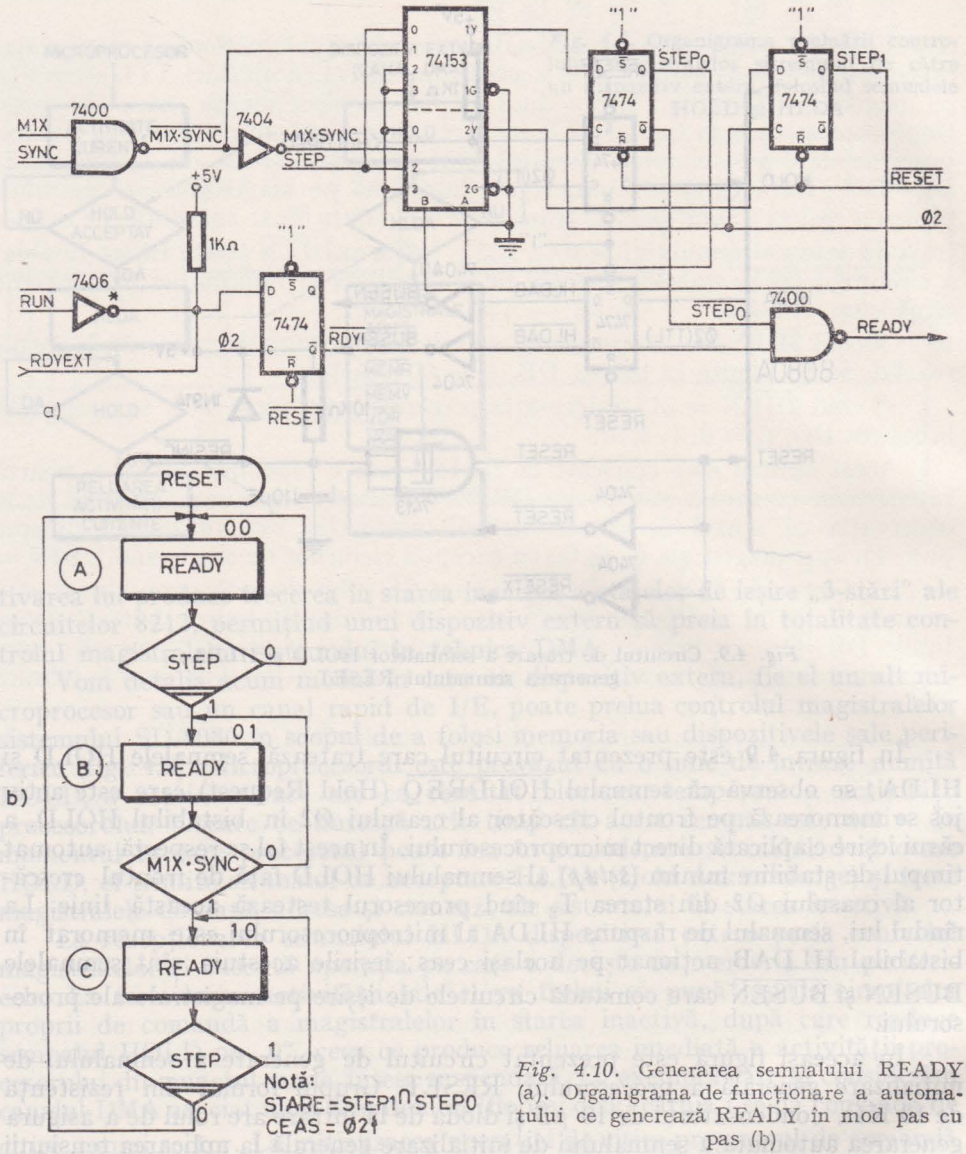


Fig. 4.10. Generarea semnalului READY (a). Organigrama de funcționare a automatului ce generează READY în mod pas cu pas (b)

Circuitul ce generează semnalul READY este prezentat în figura 4.10 *b*. Se observă că READY este dat de ecuația:

$$\text{READY} = \text{RDYI} + \text{STEP}_0,$$

unde RDYI și STEP_0 sînt create de doi bistabili de tip D acționați pe frontul pozitiv al ceasului Ø2; sincronizarea lui READY cu Ø2 asigură automat res-

pectarea timpului de stabilire minim (*setup*) al semnalului față de frontul negativ al lui Ø2. Componenta RDYI acționează pentru cazul unei derulări continue a programului forțînd semnalul READY pe „1" pe baza semnalului $\overline{RUN}$ = „0", lucru ce semnifică faptul că cheia de pe panoul de comandă este pusă în poziția DERULARE. Notăm că în cazul unei derulări normale a programului semnalul READY poate fi menținut continuu pe „1", pentru că atît memoria, cît și dispozitivele de I/E ale SD-8080 pot răspunde în timp util oricărei solicitări a procesorului. Punerea cheii în poziția STOP produce RUN = = „0" ($\overline{RUN}$ = „1"), deci prin intermediul lui RDYI obținem READY = = „0", ceea ce atrage oprirea pentru o perioadă nedeterminată a procesorului în starea WAIT. Semnalul RDYEXT permite comanda din exterior a liniei READY, trecînd prin bistabilul de sincronizare RDYI.

Componenta $STEP_0$ comandă derularea programului în mod PAS CU PAS, un pas putînd fi reprezentat de un singur ciclu-mașină, sau de ansamblul ciclurilor care alcătuiesc o instrucțiune. Această funcțiune este implementată cu ajutorul automatului format din bistabilii de tip D numiți $STEP_0$ și $STEP_1$ în conjuncție cu circuitul dublu multiplexor 4:1 de tip 74153 care creează funcția de stare următoare a automatului. Funcționarea acestuia e descrisă de organigrama din figura 4.10*b*. Se observă că automatul este inactiv în starea **A**, așteptînd apăsarea temporară a cheii de derulare în poziția PAS CU PAS, eveniment care are ca efect generarea semnalului STEP = „1", pe timpul acționării. În starea **B** se generează READY, iar durata acestei stări determină tipul pasului în funcție de poziția cheii INSTRUCȚIUNE/CICLU de pe panoul de comandă:

— în poziția INSTRUCȚIUNE semnalul M1X este conectat la M1, ieșirea corespunzătoare bitului 5 din octetul de stare, ceea ce face ca procesorul să primească READY pînă la frontul pozitiv al ceasului Ø2 din ciclul T_2 al instrucțiunii următoare, cînd condiția M1 · SYNC are ca efect tranziția de stare **B** → **C** a automatului și dispariția imediată a lui READY;

— în poziția CICLU semnalul M1X este făcut egal cu „1", deci condiția de tranziție în starea **C** a automatului este întîlnită în următoarea stare T_2 a procesorului, făcînd ca READY să cadă și procesorul să se oprească în WAIT.

În starea **C** se așteaptă dispariția semnalului STEP, pentru a evita declanșarea multiplă a automatului pe o singură apăsare a cheii de derulare.

Funcțiile de generare a semnalului RESET, de sincronizare a lui READY cu Ø2 și de generare a semnalelor de ceas în niveluri MOS pot fi mai economic realizate cu ajutorul circuitului specializat Intel 8224 - CLOCK GENERATOR. Pentru funcționarea acestui circuit este necesară conectarea în exterior a unui cuarț de 18 MHz.

4.2.3. MEMORIA

Memoria SD-8080 este organizată pe octeți, corespunzător lărgimii de 8 biți a magistralei de date. Fiecare octet are o adresă unică de 16 biți reprezentînd poziția lui secvențială în memorie. Volumul memoriei SD-8080 este de 64 Kocteți (64 K = 65 536 = 2^{16}) atingînd capacitatea maximă de adresare a microprocesorului 8080 pe cei 16 biți ai magistralei de adrese.

Din totalitatea memoriei, o parte de tip PROM (Programmable Read Only Memory) însumează un volum de 20 Kocteți, iar cealaltă de tip RAM (Random Access Memory) ocupă un volum de 44 Kocteți.

În figura 4.11 este prezentată harta memoriei SD-8080. Se observă că memoria fixă PROM e constituită din două blocuri, primul cu lungimea de 4 Kocteți ocupă adresele de la 0000_{16} la $0FFF_{16}$, iar celălalt ocupă 16 Kocteți

		ADRESA (HEX)
PROM 4K (3601)	MONITOR ȘI TRATARE ÎNTRERUPERI	0000
	DRIVER-I PERIFERICE: CI, CØ, PØ, RI, TI, TØ	0697
	PTRAM	0703
	PTROM	0816
	TABELUL DE SEMNĂTURI CIRCUITE PROM	0900
		0F00
		0F40
RAM 44K (12K×2102+32K×F16K)	ZONA DE LUCRU INTERPRETOR BASIC	1000
	ZONA DE LEGĂTURI ALE INTERPRETORULUI BASIC	1200
	RUTINE *ASSEMBLER* DEFINITE DE UTILIZATOR ALE INTERPRETORULUI BASIC	1240
	ZONA DE LEGĂTURI ALE INTERPRETORULUI CU RUTINELE *ASSEMBLER*	31E0
	PROGRAM SURSĂ BASIC, TABELE, VARIABILE	3240
	STIVA INTERPRETORULUI BASIC	BDD3
	STIVA MONITORULUI (32 OCT)	BDF3
	ZONA DE SALVARE A MONITORULUI (13 OCT)	BE00
	ZONA DE LUCRU PENTRU PERIFERICE ȘI ÎNTRERUPERI	BEF4
	ZONA DE ARGUMENTE ALE *DRIVER*-ILOR DE CASETĂ (ZONA F=7 OCT+3 OCT)	BEFE
	BUFFER-UL DE I/E AL *DRIVER*-ILOR DE CASETĂ (258 OCT)	C000
REPROM 16K (2708)	ZONA DE LEGĂTURI ALE MONITORULUI	C040
	INTERPRETORUL BASIC	E380
	BIBLIOTECA ARITMETICĂ A INTERPRETORULUI BASIC	F400
	DRIVER-I DE CASETĂ	

Fig. 4.11. Harta memoriei SD-8080.

între adresele $C000_{16}$ și $FFFF_{16}$. Memoria cu conținut aleator RAM formează un segment contiguu de 44 Kocteți plasat între adresele 1000_{16}—$BFFF_{16}$; la rîndul ei, este formată din două blocuri: între adresele 1000_{16} și $3FFF_{16}$ se găsesc primii 12 Kocteți implementați cu circuite de memorie RAM statică, iar următorii 32 Kocteți aflați între adresele 4000_{16} și $BFFF_{16}$ sînt implementați cu circuite de memorie RAM dinamice. Pe figură este indicat, de asemenea, și modul de amplasare a programelor în memorie și întinderea lor: de exemplu monitorul SD-8080 și programul de tratare a întreruperilor ocupă primul segment al memoriei cuprins între adresele 000 și 696_{16}.

Vom descrie în continuare, din punct de vedere constructiv, cele patru module (blocuri) ale memoriei, întrucît fiecare bloc este implementat cu un alt tip de circuit și de aceea poate ilustra anumite particularități de proiectare.

4.2.3.1. Memoria PROM implementată cu circuite 3601

Segmentul de memorie fixă aflat între adresele 000-FFF_{16} (4 Kocteți) este realizat cu circuite bipolare Schottky de tip Intel 3601. Caracteristicile principale ale acestor circuite sînt:

— organizare: 256 cuvinte × 4 biți;
— capsulă: 16 pini;
— alimentare: $+5$ V $\pm$ 5%;
— timp maxim de acces: 70 ns;
— intrări și ieșiri în niveluri TTL;
— bitul neprogramat se prezintă la ieșire ca un „0" logic;
— 8 intrări de adresă A_0—A_7; decodificatoare interne;
— 4 ieșiri de date „colector în gol" O_1—O_4;
— 2 intrări de selecție active jos $\overline{CE_1}$, $\overline{CE_2}$; dacă cel puțin una dintre ele e la „1" tranzistorii de pe ieșirile de date sînt blocați;
— folosirea conexiunilor fuzibile de bit din siliciu policristalin îi conferă o foarte bună siguranță în funcționare; în literatură este menționat un test efectuat asupra acestui tip de circuit în care după 3×10^{12} conexiuni·ore de funcționare la 85°C nu s-a înregistrat nici un defect.

Pentru realizarea unui Koctet de memorie sînt necesare 8 asemenea circuite. În figura 4.13 este prezentată implementarea primului Kilooctet de PROM al SD-8080 situat în spațiul de adrese 000-$3FF_{16}$. Aceeași schemă este valabilă și pentru fiecare dintre ceilalți Kilococteți de PROM realizați cu circuite 3601, înlocuind semnalul de selecție $\overline{MS_0}$ (Memory Select), activ jos, cu $\overline{MS_1}$, $\overline{MS_2}$ și, respectiv, cu $\overline{MS_3}$. Semnalele $\overline{MS_i}$ ($i = 0, 1, 2, ..., 15$) identifică al $i + 1$-lea Kilooctet din primii 16 Kocteți ai memoriei (fig. 4.12), pe baza decodificării liniilor de adresă $ABUS_{13}$, $ABUS_{12}$, $ABUS_{11}$ și $ABUS_{10}$. Pe schema din figura 4.13 semnalul $\overline{MS_0}$ atacă intrarea de validare $\overline{CE_1}$ a tuturor circuitelor 3601 aparținînd primului Kilooctet al memoriei. Semnalele $\overline{ROMS_i}$ (ROM Select), $i = 0, 1, 2, 3$, active jos, reprezentînd decodificarea liniilor de adresă $ABUS_9$ și $ABUS_8$ (fig. 4.12), selectează numai unul

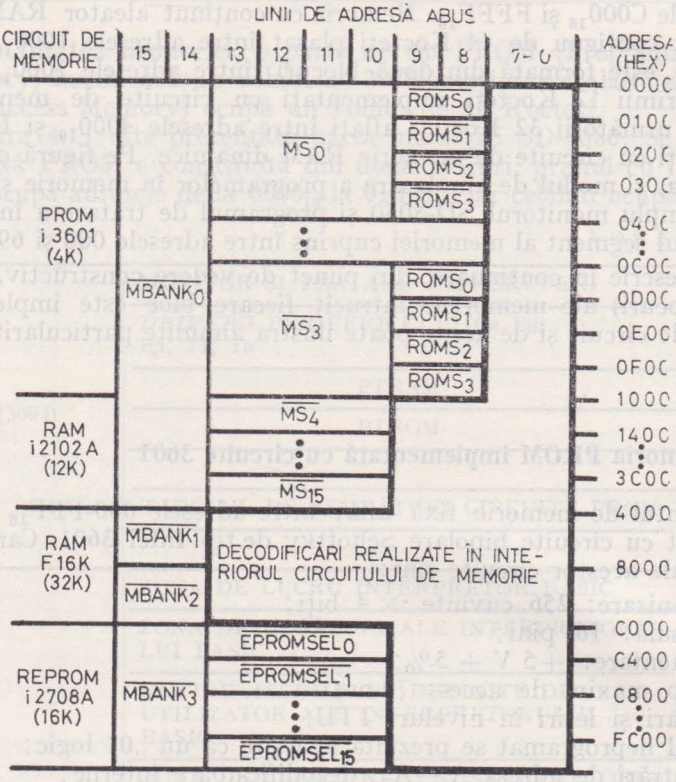


Fig. 4.12. Harta adreselor de memorie și a decodificării conținutului magistralei de adrese în vederea obținerii semnalelor de selecție

dintre cele patru grupuri de cîte două circuite 3601 corespunzînd fiecare unui sfert de Kilooctet (256 octeți), fiind legate pe intrările $\overline{CE_2}$ ale circuitelor de memorie.

Rezultă așadar că se folosește o schemă de selecție bidimensională (2D) care face ca atunci cînd se efectuează un acces la o adresă cuprinsă între 000-FFF_{16} să se selecteze un singur grup de două circuite 3601 și anume acela pentru care se îndeplinește condiția:

$$\overline{MS_i} + \overline{ROMS_j} = \text{„0“},$$

unde $i = 0, 1, 2, 3$ și $j = 0, 1, 2, 3$.

Selectarea uneia dintre cele 256 de adrese interne din circuit cade în sarcina decodificatoarelor interne ale lui 3601; pentru aceasta este suficientă conectarea directă a intrărilor A_7—A_0 ale circuitelor respectiv la liniile magistralei de adrese $ABUS_7$—$ABUS_0$. Ieșirile circuitelor 3601 sînt conectate direct pe magistrala de date ale cărei linii sînt prevăzute cu rezistențe de polarizare de 1KΩ (fig. 4.7*b*). Dintre cele două circuite 3601 care sînt simultan

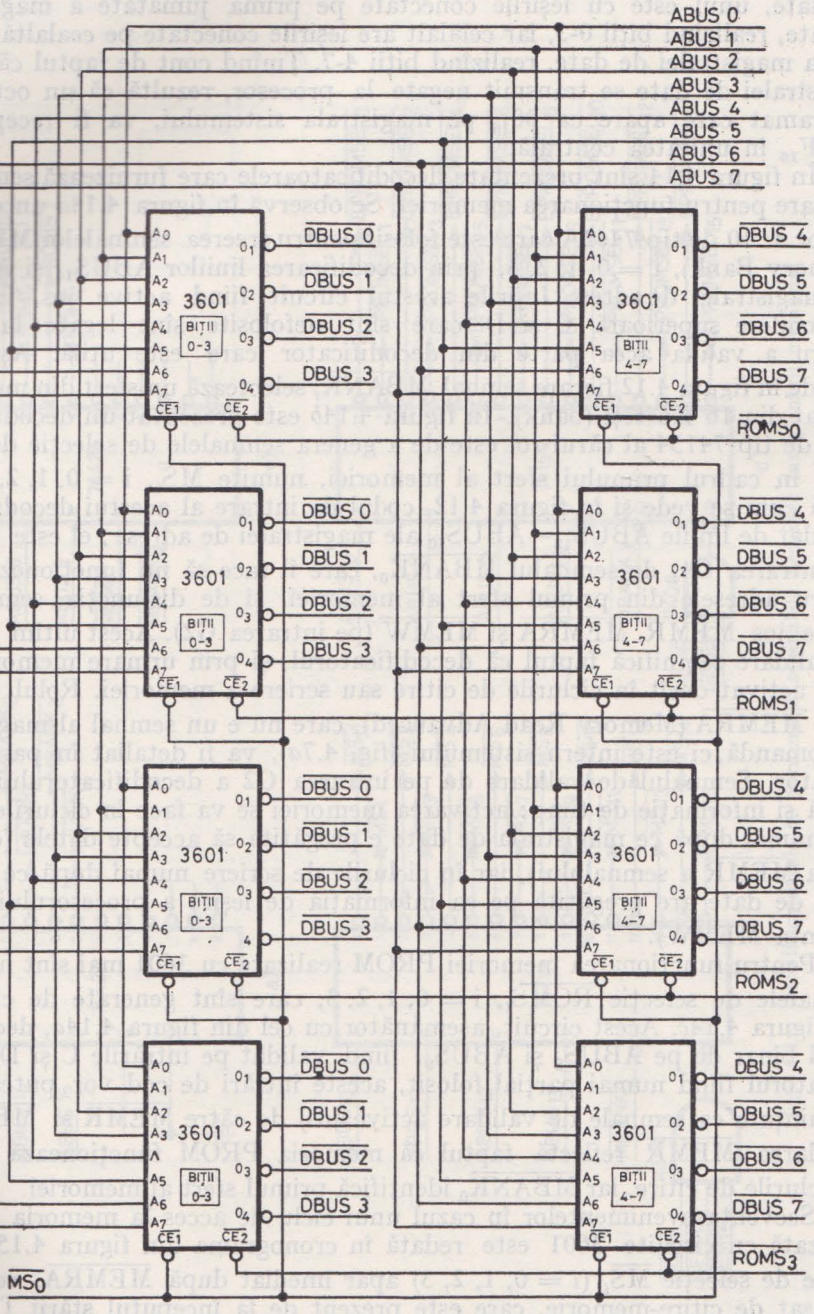


Fig. 4.13. Implementarea cu circuite 3601 a unui Kilooctet de PROM

selectate, unul este cu ieșirile conectate pe prima jumătate a magistralei de date, realizînd biții 0-3, iar celălalt are ieșirile conectate pe cealaltă jumătate a magistralei de date, realizînd biții 4-7. Ținînd cont de faptul că liniile magistralei de date se transmit negate la procesor, rezultă că un octet neprogramat care apare ca 00_{16} pe magistrala sistemului, va fi recepționat ca FF_{16} în unitatea centrală.

În figura 4.14 sînt prezentate decodificatoarele care furnizează semnalele necesare pentru funcționarea memoriei. Se observă în figura 4.14*a* un decodificator 4: 10 de tip 7442A care este folosit pentru creerea semnalelor $\overline{MBANK}_i$ (Memory Bank), i = 0, 1, 2, 3, prin decodificarea liniilor $ABUS_{14}$ și $ABUS_{15}$ ale magistralei de adrese. Ieșirile acestui circuit fiind active jos, intrările de pondere superioară C și D care sînt nefolosite sînt legate la masă pentru a valida acea parte din decodificator care este utilă. Așa cum se vede în figura 4.12 fiecare semnal $\overline{MBANK}_i$ selectează un sfert din memorie, format din 16 Kocteți (bank). În figura 4.14*b* este prezentat un decodificator 4:16 de tip 74154 al cărui rol este de a genera semnalele de selecție de Kilooctet în cadrul primului sfert al memoriei, numite $\overline{MS}_i$, i = 0, 1, 2, ... ,15. După cum se vede și în figura 4.12, codul de intrare al acestui decodificator este dat de liniile $ABUS_{13}$—$ABUS_{10}$ ale magistralei de adrese; el este validat (pe intrarea G1) de semnalul $\overline{MBANK}_0$, care îl face să nu funcționeze decît pentru adresele din primul sfert al memoriei, și de disjuncția semnalelor active jos $\overline{MEMR}$, $\overline{MEMRA}$ și $\overline{MEMW}$ (pe intrarea G2). Acest ultim semnal de validare semnifică faptul că decodificatorul, și prin urmare memoria, nu va fi activat decît în ciclurile de citire sau scriere a memoriei. Rolul semnalului $\overline{MEMRA}$ (Memory Read Advanced), care nu e un semnal al magistralei de comandă, ci este intern sistemului (fig. 4.7*a*), va fi detaliat în paragraful următor. Semnalul de validare de pe intrarea G2 a decodificatorului înglobează și informație de timp: activarea memoriei se va face în ciclurile de citire numai după ce magistrala de date e pregătită să accepte datele (componenta $\overline{MEMR}$ a semnalului), iar în ciclurile de scriere numai după ce magistrala de date are pregătită pe ea informația de ieșire a procesorului (componenta $\overline{MEMW}$).

Pentru funcționarea memoriei PROM realizate cu 3601 mai sînt necesare semnalele de selecție $\overline{ROMS}_i$, i = 0, 1, 2, 3, care sînt generate de circuitul din figura 4.14*c*. Acest circuit, asemănător cu cel din figura 4.14*a*, decodifică codul binar de pe $ABUS_8$ și $ABUS_9$, fiind validat pe intrările C și D (decodificatorul fiind numai parțial folosit, aceste intrări de cod vor putea fi întrebuințate ca semnale de validare active jos) de către $\overline{MEMR}$ și $\overline{MBANK}_0$. Validarea $\overline{MEMR}$ reflectă faptul că memoria PROM funcționează numai în ciclurile de citire, iar $\overline{MBANK}_0$ identifică primul sfert al memoriei.

Secvența evenimentelor în cazul unui ciclu de acces la memoria PROM realizată cu circuite 3601 este redată în cronograma din figura 4.15. Semnalele de selecție $\overline{MS}_i$ (i = 0, 1, 2, 3) apar imediat după $\overline{MEMRA}$, semnalul avansat de citire-memorie, care este prezent de la începutul stării T_2; citirea începe deci practic odată cu apariția lui $\overline{ROMS}_i$ (i = 0, 1, 2, 3) care este

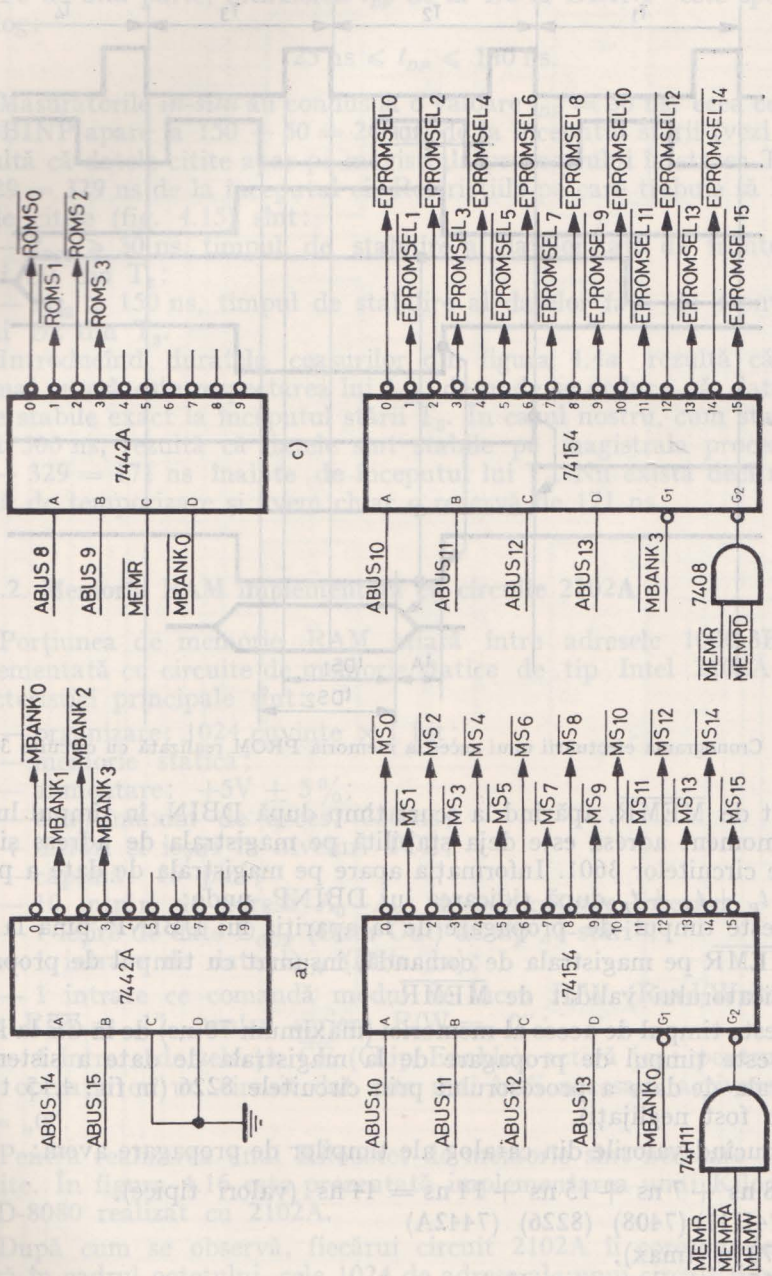


Fig. 4.14. Decodificatoarele ce furnizează semnalele necesare funcționării memoriei

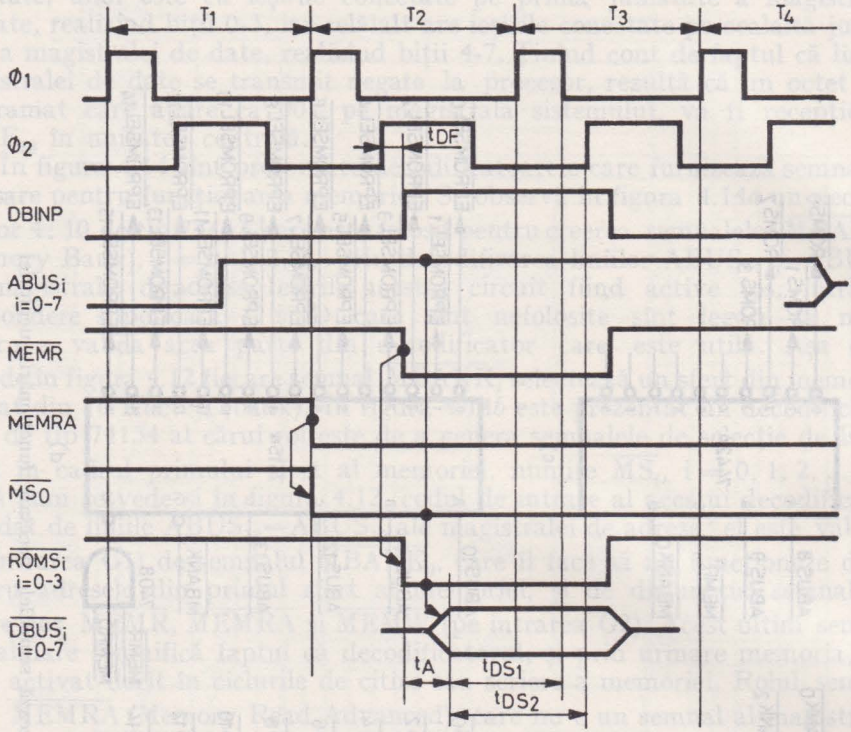


Fig. 4.15. Cronograma efectuării unui acces la memoria PROM realizată cu circuite 3601

condiționat de $\overline{\text{MEMR}}$, apărînd la scurt timp după DBIN, în timpul lui T_2. La acest moment adresa este deja stabilită pe magistrala de adrese și deci la intrările circuitelor 3601. Informația apare pe magistrala de date a procesorului la $t_P + t_A + t_B$ după ridicarea lui DBINP, unde:

— t_P este timpul de propagare de la apariția lui DBINP pînă la apariția lui $\overline{\text{MEMR}}$ pe magistrala de comandă, însumat cu timpul de propagare al decodificatorului validat de $\overline{\text{MEMR}}$;

— t_A este timpul de acces al memoriei (maximum 70 ns) de la $\overline{\text{CE}}$ la ieșiri;

— t_B este timpul de propagare de la magistrala de date a sistemului la magistrala de date a procesorului prin circuitele 8226 (în fig. 4.15 timpii t_P și t_B au fost neglijați).

Introducînd valorile din catalog ale timpilor de propagare avem:

$t_P = 8$ ns $+ 7$ ns $+ 15$ ns $+ 14$ ns $= 44$ ns (valori tipice),
(74H11) (7408) (8226) (7442A)
$t_A = 70$ ns (max),
$t_B = 15$ ns (valoare tipică)

și deci:

$$t_P + t_A + t_B = 44 + 70 + 15 = 129 \text{ ns}.$$

Pe de altă parte, întîrzierea t_{DF} de la Ø2 la DBINP este specificată în catalog:

$$25 \text{ ns} \leqslant t_{DF} \leqslant 140 \text{ ns}.$$

Măsurătorile *in-situ* au condus la o valoare $t_{DF} = 50$ ns, ceea ce înseamnă că DBINP apare la 150 + 50 = 200 ns de la începutul stării (vezi fig. 4.4*a*). Rezultă că datele citite apar pe magistrala procesorului în starea T_2 la 200 + + 129 = 329 ns de la începutul ei. Restricțiile pe care trebuie să le respecte datele citite (fig. 4.15) sînt:

— $t_{DS1} \geqslant 30$ ns, timpul de stabilire al datelor față de frontul negativ al lui Ø1 din T_3;

— $t_{DS2} \geqslant 150$ ns, timpul de stabilire al datelor față de frontul pozitiv al lui Ø2 din T_3.

Introducînd duratele ceasurilor din figura 4.4*a* rezultă că restricția cea mai severă este respectarea lui t_{DS2}, de unde se deduce că datele trebuie să fie stabile exact la începutul stării T_3. În cazul nostru, cum starea T_2 durează 500 ns, rezultă că datele sînt stabile pe magistrala procesorului cu 500 — 329 = 171 ns înainte de începutul lui T_3. Nu există deci nici o problemă de temporizare și avem chiar o rezervă de 171 ns.

4.2.3.2. Memoria RAM implementată cu circuite 2102A

Porțiunea de memorie RAM aflată între adresele $1000\text{-}3FFF_{16}$ este implementată cu circuite de memorie statice de tip Intel 2102A, ale căror caracteristici principale sînt:

— organizare: 1024 cuvinte × 1 bit;
— memorie statică;
— alimentare: +5V ± 5%;
— timp maxim de acces: 350 ns;
— intrări și ieșiri în niveluri TTL;
— capsulă: 16 pini;
— 10 intrări de adresă A_0—A_9; decodificatoare interne;
— 1 ieșire de date D_{OUT} (Data Out) de tip „3-stări";
— 1 intrare de date D_{IN} (Data In);
— 1 intrare ce comandă modul de lucru R/W (Read/Write); pentru citire R/W = „1", pentru scriere R/W = „0";
— 1 intrare de selecție $\overline{CE}$ (Chip Enable), activă jos; pentru a efectua orice operație cu un circuit dat este mai întîi necesară activarea lui prin $\overline{CE}$ = „0".

Pentru realizarea unui Kilooctet de memorie sînt necesare 8 asemenea circuite. În figura 4.16 este prezentată implementarea unui Kilooctet RAM al SD-8080 realizat cu 2102A.

După cum se observă, fiecărui circuit 2102A îi corespunde o poziție binară în cadrul octetului, cele 1024 de adrese ale unui circuit fiind selectate prin conectarea directă a intrărilor A_0—A_9 ale circuitului la liniile $ABUS_0$—$ABUS_9$ ale magistralei de adrese. Fiecare grup de 8 circuite 2102A consti-

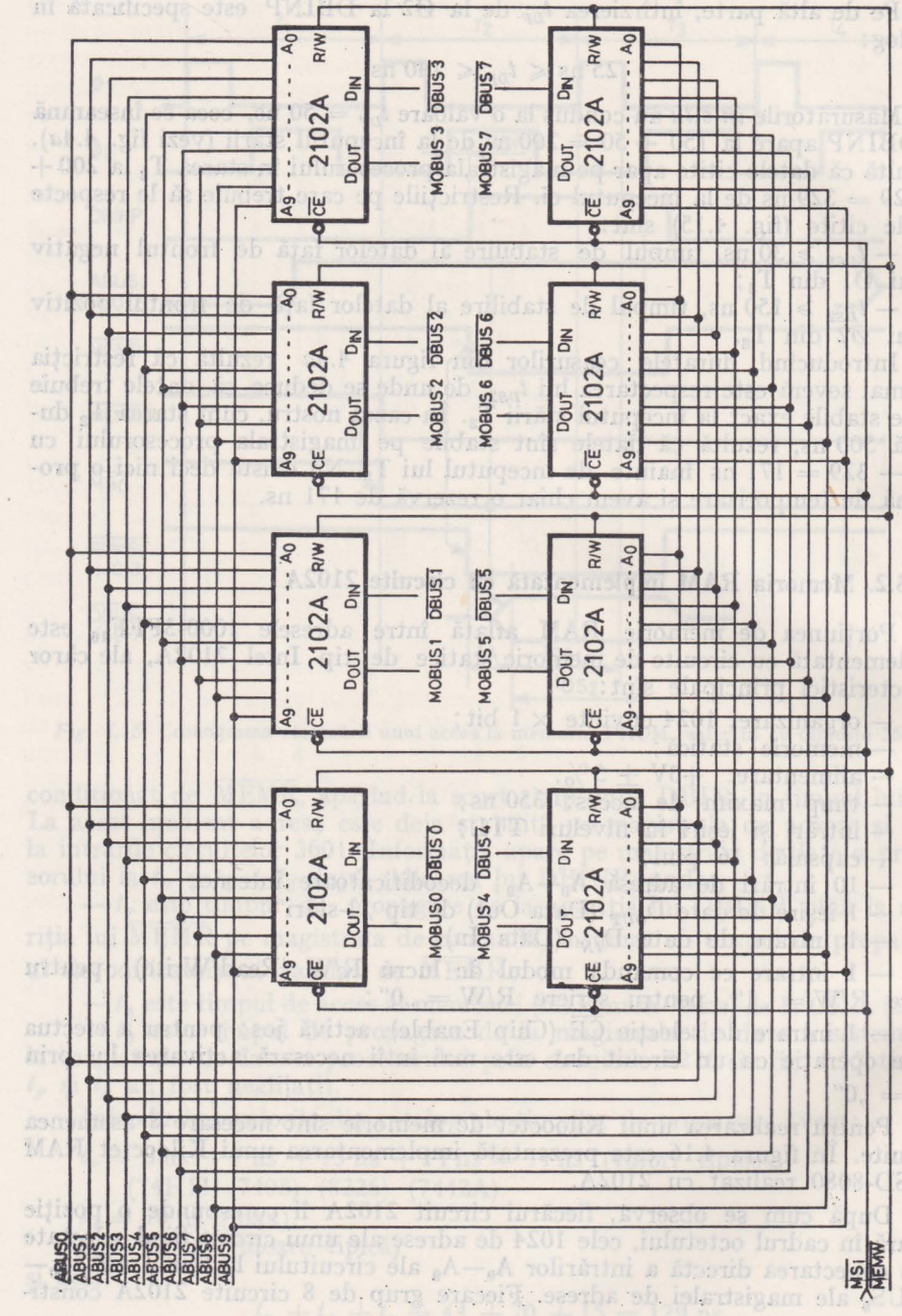


Fig. 4.16. Implementarea cu circuite 2102A a unui Kilooctet de memorie RAM

tuind un Kilooctet este activat pe intrarea $\overline{CE}$ cu ajutorul unui semnal $\overline{MS_i}$, $i = 4, 5, \ldots, 15$ conform hărții adreselor și semnalelor de selecție ale memoriei prezentate în figura 4.12. Aceste semnale sînt produse de decodificatorul arătat în figura 4.14*b*, a cărui funcționare a fost prezentată în paragraful anterior. Intrările de mod de funcționare R/W ale tuturor circuitelor 2102A sînt conectate la semnalul $\overline{MEMW}$; dacă atunci cînd memoria e selectată $\overline{MEMW}$ = „0" înseamnă că se dorește o scriere, în caz contrar este vorba de o citire.

În timpul unui ciclu de scriere a memoriei circuitul 2102A primește datele pe care le memorează prin intermediul intrării D_{IN}, în cazul nostru fiecare din cele 8 intrări D_{IN} este conectată la cîte o linie a magistralei de date a sistemului $\overline{DBUS_0}$—$\overline{DBUS_7}$. Remarcăm că în memorie informația va fi introdusă în formă complementată. În cazul unui ciclu de citire informația citită va fi regăsită pe ieșirile D_{OUT} ale celor 8 circuite care formează un Kilooctet de memorie; toate ieșirile corespunzînd unui același bit din cei 12 Kocteți de RAM realizați cu 2102A fiind de tip „3-stări", ele sînt conectate împreună pe o linie internă $MOBUS_i$ (Memory Output Bus), $i = 0, 1, 2, \ldots, 7$. Sub acțiunea semnalelor de selecție, numai o singură ieșire din cele 12 va fi la un moment dat activă pe orice linie $MOBUS_i$, aceea corespunzînd Kilooctetului adresat. Pe de altă parte, întrucît circuitele 2102A nu pot absorbi pe ieșirea D_{OUT} decît 2,1 mA în starea joasă, liniile $MOBUS_i$ nu pot fi conectate pentru a conduce direct magistrala de date. Din acest motiv a fost necesară introducerea unui circuit de ieșire al memoriei RAM realizate cu 2102A, care reproduce informația de pe magistrala internă a memoriei pe magistrala de date a sistemului. Acest circuit, prezentat în figura 4.17, este constituit dintr-un 8212 conectat în modul de intrare (MD = „0", $\overline{CLR}$ = STB = „1") primind pe liniile $DI_1 - DI_8$ semnalele magistralei interne a memoriei $MOBUS_0 - MOBUS_7$ și fiind conectat cu ieșirile sale $DO_1 - DO_8$ la magistrala de date a sistemului $\overline{DBUS_0} - \overline{DBUS_7}$. Întrucît acest circuit nu trebuie să-și activeze ieșirile decît în ciclurile de citire a memoriei, cînd magistrala de date introduce informație în procesor, intrarea sa de selecție $\overline{DS_1}$ e conectată la $\overline{MEMR}$, iar DS_2 la semnalul compus RAMSEL, unde:

$$RAMSEL = \overline{(MS_0 + MS_1 + MS_2 + MS_3)} \cdot MBANK_0.$$

În ecuația de mai sus termenul $\overline{MS_0 + MS_1 + MS_2 + MS_3}$ identifică situațiile cînd nu se adresează vreunul din primii 4 Kocteți ai memoriei, iar al doilea ne asigură că adresa se referă la o locație dintre primii 16 Kocteți.

Secvența evenimentelor în cazul unui ciclu de adresare a acestui tip de memorie e prezentată în cronograma din figura 4.18. În situația unui ciclu de citire circuitele de memorie ce corespund Kilooctetului adresat își încep activitatea odată cu apariția semnalului de selecție $\overline{MS_i}$ ($i = 4, 5, \ldots, 15$) corespunzător, care se poziționează datorită semnalului avansat de citire-memorie, $\overline{MEMRA}$. Acesta are rolul de a lansa ciclul de citire în interiorul memoriei, chiar dacă la momentul apariției lui, începutul stării T_2, magis-

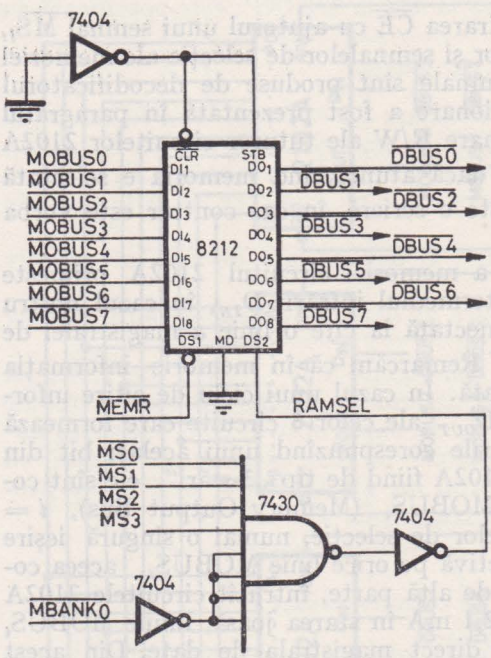


Fig. 4.17. Circuitul de ieşire al memoriei RAM realizate cu 2102A

trala de date nu poate accepta datele citite din memorie. Folosirea lui, ca şi introducerea circuitului de ieşire din figura 4.17, permite lansarea anticipată a ciclului de citire a memoriei şi evitarea totală a stării WAIT în cazul lucrului cu acest tip de memorie.

Momentul obţinerii informaţiei stabile pe magistrala de date a procesorului, măsurat de la începutul stării T_2, este:

$$\max(t_1, t_2) + t_B, \text{ unde:}$$

— t_1 este întîrzierea calculată pe calea de propagare: $\overline{\text{MEMRA}}$, $\overline{\text{MS}_i}$, selecţie 2102A, stabilire $\overline{\text{DBUS}_i}$ $(i = 0, 1, \ldots, 7)$;

— t_2 reprezintă întîrzierea pe calea: DBINP, $\overline{\text{MEMR}}$, trecerea datelor de la MOBUS_i la $\overline{\text{DBUS}_i}$ $(i = 0, 1, 2, \ldots, 7)$;

— t_B este timpul de propagare de la magistrala de date a sistemului la magistrala de date a procesorului prin circuitele 8226.

Avem atunci:

$$t_1 = \underset{(8212)}{30 \text{ ns}} + \underset{(7404)}{8 \text{ ns}} + \underset{(74H11)}{9 \text{ ns}} + \underset{(74154)}{18 \text{ ns}} + \underset{(2102A)}{180 \text{ ns}} + \underset{(8212)}{30 \text{ ns}} = 275 \text{ ns}$$

şi respectiv:

$$t_2 = \underset{(t_{\varnothing 1 \varnothing 2})}{150 \text{ ns}} + \underset{(t_{DF})}{50 \text{ ns}} + \underset{(74H11)}{8 \text{ ns}} + \underset{(7408)}{7 \text{ ns}} + \underset{(8226)}{15 \text{ ns}} + \underset{(8212)}{30 \text{ ns}} = 260 \text{ ns},$$

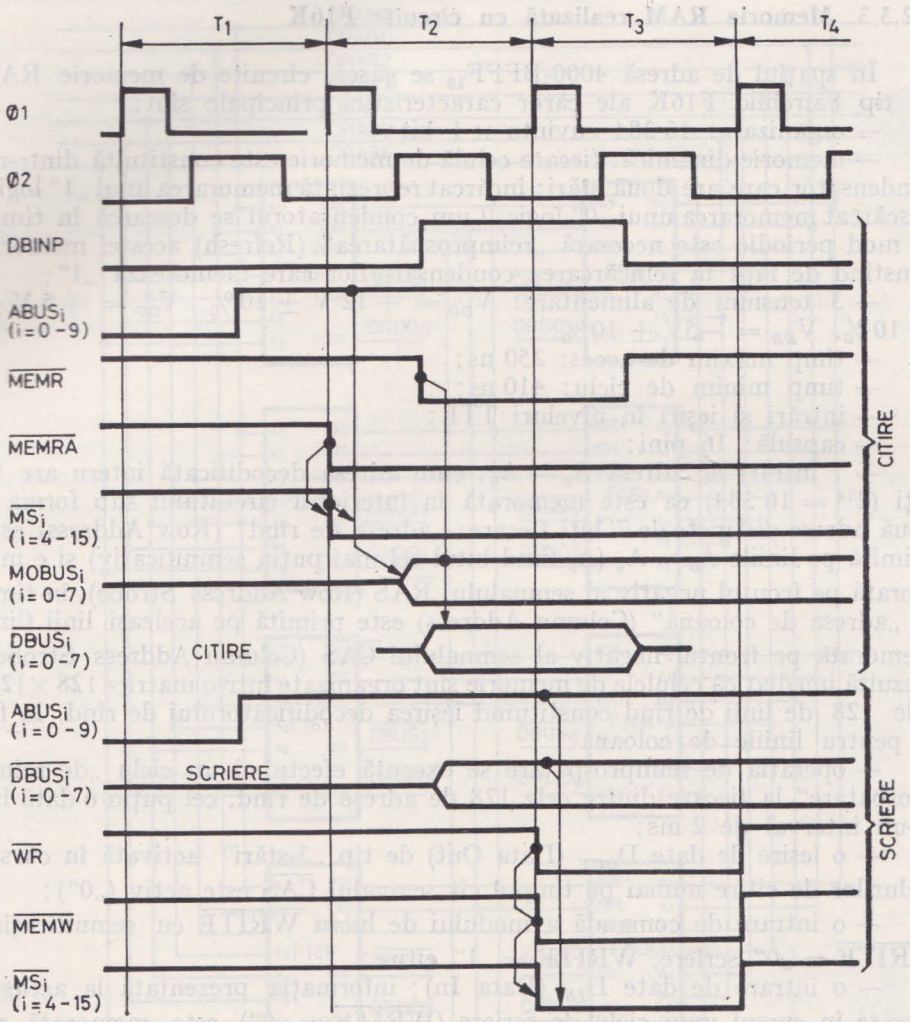


Fig. 4.18. Cronograma unui ciclu de acces la memoria RAM realizată cu circuite 2102A

de unde rezultă că timpul la care se obține informația stabilă pe magistrala procesorului este $275 + 15 = 290$ ns după ridicarea lui Ø1 din T_2, ceea ce satisface cu prisosință restricțiile impuse și justifică aserțiunea READY = „1" pentru acest tip de memorie.

În privința ciclului de scriere în memoria realizată cu 2102A, tot pe figura 4.18 se poate observa că la apariția lui $\overline{WR}$, care generează $\overline{MEMW}$ și $\overline{MS_i}$ ($i = 4, 5, \ldots, 15$), intrările de date și de adresă ale circuitelor 2102A sînt stabile, întrunindu-se toate condițiile de timp necesare pentru buna funcționare la scriere a circuitului 2102A.

4.2.3.3. Memoria RAM realizată cu circuite F16K

În spațiul de adresă 4000-$BFFF_{16}$ se găsesc circuite de memorie RAM de tip Fairchild F16K ale căror caracteristici principale sînt:

— organizare: 16 384 cuvinte × 1 bit;

— memorie dinamică: fiecare celulă de memorie este constituită dintr-un condensator care are două stări: încărcat reprezintă memorarea unui „1" logic, descărcat memorarea unui „0" logic. Cum condensatorul se descarcă în timp, în mod periodic este necesară „reîmprospătarea" (Refresh) acestei memorii, constînd de fapt în reîncărcarea condensatorilor care memorează „1";

— 3 tensiuni de alimentare: $V_{DD} = +12\ V \pm 10\%$; $V_{CC} = +5\ V \pm \pm 10\%$, $V_{BB} = -5V \pm 10\%$;

— timp maxim de acces: 250 ns;

— timp minim de ciclu: 410 ns;

— intrări și ieșiri în niveluri TTL;

— capsulă: 16 pini;

— 7 intrări de adresă $A_0 - A_6$; cum adresa decodificată intern are 14 biți ($2^{14} = 16\ 384$) ea este memorată în interiorul circuitului sub forma a două adrese distincte de 7 biți fiecare: „adresa de rînd" (Row Address) este primită pe liniile $A_0 - A_6$ (A_0 fiind bitul cel mai puțin semnificativ) și e memorată pe frontul negativ al semnalului $\overline{RAS}$ (Row Address Strobe), în timp ce „adresa de coloană" (Column Address) este primită pe aceleași linii fiind memorată pe frontul negativ al semnalului $\overline{CAS}$ (Column Address Strobe). Rezultă imediat că celulele de memorie sînt organizate într-o matrice 128 × 128, cele 128 de linii de rînd constituind ieșirea decodificatorului de rînd, la fel și pentru liniile de coloană;

— operația de reîmprospătare se execută efectuînd un ciclu „de reîmprospătare" la fiecare dintre cele 128 de adrese de rînd, cel puțin o dată într-un interval de 2 ms;

— o ieșire de date D_{OUT} (Data Out) de tip „3-stări" activată în cursul ciclurilor de citire numai pe timpul cît semnalul $\overline{CAS}$ este activ („0");

— o intrare de comandă a modului de lucru $\overline{WRITE}$ cu semnificația: $\overline{WRITE}$ = „0" scriere, $\overline{WRITE}$ = „1" citire;

— o intrare de date D_{IN} (Data In); informația prezentată la această intrare în cursul unui ciclu de scriere ($\overline{WRITE}$ = „0") este memorată pe frontul negativ care survine ultimul dintre tranzițiile semnalelor $\overline{CAS}$ și $\overline{WRITE}$, aceasta în timp ce $\overline{RAS}$ este activ („0");

— aplicarea tensiunilor de alimentare trebuie făcută astfel ca $V_{BB} = -5$ V să fie conectată prima și deconectată ultima.

După cum se poate observa în figura 4.19, pentru realizarea a 16 Kocteți de memorie RAM sînt necesare 8 circuite F16K, fiecare dintre ele realizînd funcția de memorare pentru unul dintre cei 8 biți ai magistralei de date. Intrările de date D_{IN} ale circuitelor sînt legate direct pe magistrala de date a sistemului, în timp ce ieșirile lor D_{OUT} sînt conectate pe magistrala internă de ieșire a memoriei dinamice. Remarcăm că se memorează informația negată, la fel ca și în cazul circuitelor 2102A. Fiecare linie $DRDO_i$

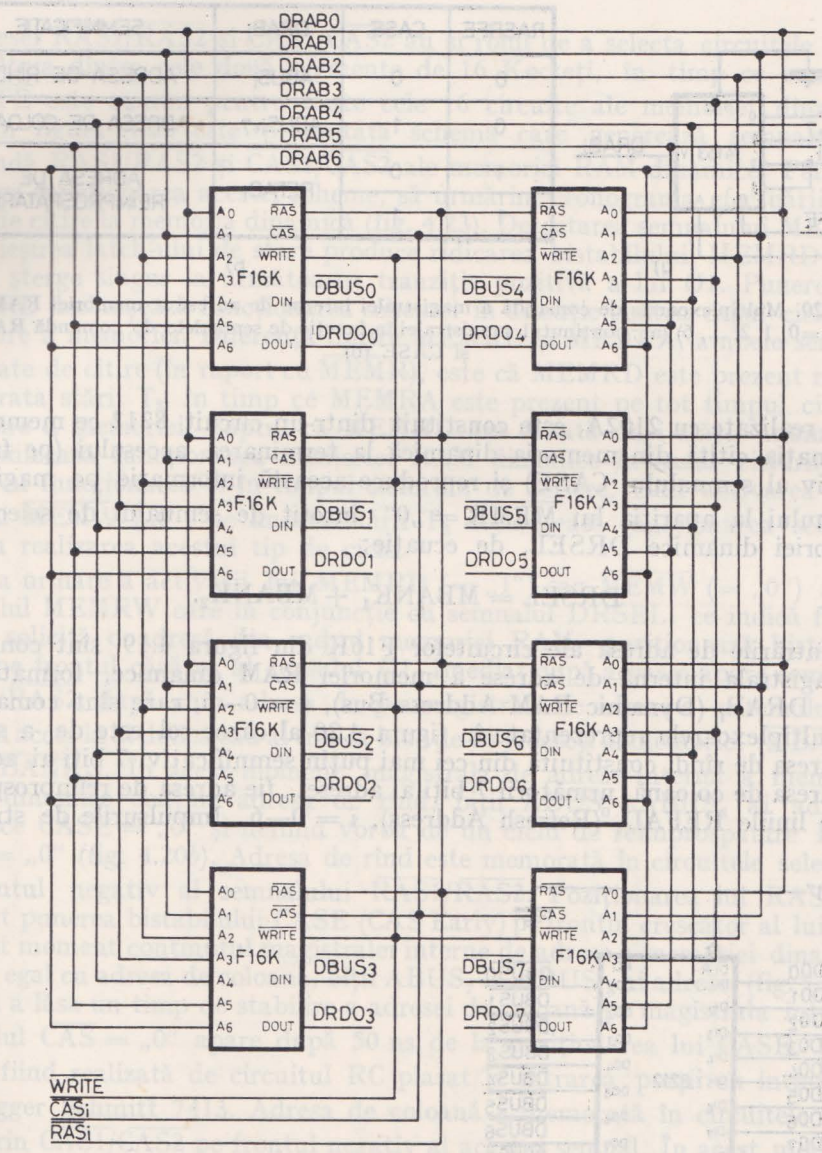
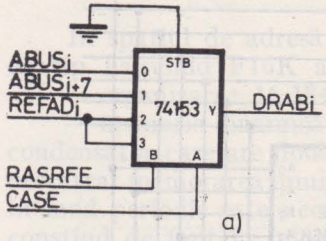


Fig. 4.19. Implementarea a 16 Kocteți de memorie RAM folosind circuite F16K

(Dinamic RAM Data Output), $i = 0 - 7$, suportă cele două ieșiri D_{OUT} ale circuitelor F16K ce implementează bitul de rang i în cele două segmente de memorie de 16 Koct în discuție și conduce intrarea corespunzătoare a circuitului de ieșire al memoriei RAM dinamice (fig. 4.21); acesta din urmă, cu un rol asemănător cu cel al circuitului de ieșire al memoriei



RASRFE	CASE	DRABi	SEMNIFICAȚIE
0	0	ABUSi	ADRESA DE RÎND
0	1	ABUSi+7	ADRESA DE COLOANĂ
1	0	REFADi	ADRESA DE REÎMPROSPĂTARE
1	1		

b)

Fig. 4.20. Multiplexoarele de comandă a magistralei interne de adrese a memoriei RAM dinamice (i=0, 1, 2, ..., 6) (a); conținutul magistralei în funcție de semnalele de comandă RASRFE și CASE (b)

RAM realizate cu 2102A, este constituit dintr-un circuit 8212 ce memorează informația citită din memoria dinamică la terminarea accesului (pe frontul negativ al semnalului CASE) și reproduce această informație pe magistrala sistemului la apariția lui $\overline{MEMR} = \text{„0"}$ însoțit de semnalul de selecție al memoriei dinamice DRSEL, de ecuație:

$$DRSEL = MBANK_1 + MBANK_2.$$

Intrările de adresă ale circuitelor F16K din figura 4.19 sînt conectate la magistrala internă de adrese a memoriei RAM dinamice, formată din liniile $DRAB_i$ (Dynamic RAM Address Bus), $i = 0—6$, care sînt comandate de multiplexoarele reprezentate în figura 4.20 al căror rol este de a selecta fie adresa de rînd, constituită din cei mai puțin semnificativi 7 biți ai adresei, fie adresa de coloană, următorii 7 biți ai adresei, fie adresa de reîmprospătare pe de liniile $REFAD_i$ (Refresh Address), $i = 0—6$. Impulsurile de strobare

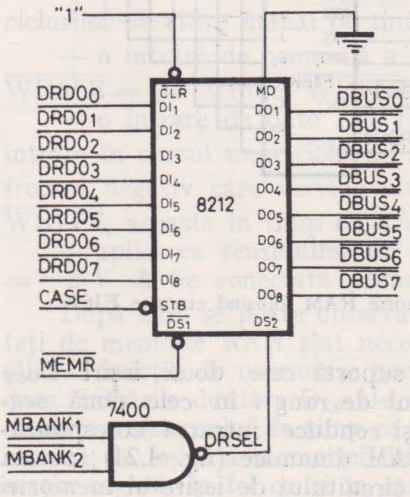


Fig. 4.21. Circuitul de ieșire al memoriei RAM dinamice

a adresei $\overline{RAS1}/\overline{RAS2}$ și $\overline{CAS1}/\overline{CAS2}$ au și rolul de a selecta circuitele unuia sau altuia dintre cele două segmente de 16 Kocteți, în timp ce semnalul $\overline{WRITE}$ este comun pentru toate cele 16 circuite ale memoriei dinamice.

În figura 4.22 este prezentată schema care generează semnalele de comandă $\overline{RAS1}/\overline{RAS2}$ și $\overline{CAS1}/\overline{CAS2}$ ale memoriei RAM dinamice. Pentru a înțelege funcționarea acestei scheme, să urmărim cronograma efectuării unui ciclu de citire la memoria dinamică (fig. 4.23). Detectarea semnalului MEMRA de la ieșirea latch-ului de stare produce ridicarea bistabilului MEMRD care se va șterge singur la următoarea tranziție pozitivă a lui Ø1. Punerea lui MEMRD declanșează funcționarea schemei în discuție pentru cazul ciclurilor de citire a memoriei. Diferența dintre MEMRD și MEMRA, ambele semnale avansate de citire (în raport cu $\overline{MEMR}$), este că MEMRD este prezent numai pe durata stării T_2, în timp ce MEMRA este prezent pe tot timpul ciclului de citire a memoriei. Faptul că MEMRD are durata unei stări, anume T_2, uniformizează funcționarea memoriei RAM dinamice în cazul ciclurilor de citire cu funcționarea ei în timpul ciclurilor de scriere, cînd activarea semnalului $\overline{MEMW}$ („0") pe durata stării T_3 declanșează funcționarea schemei pentru realizarea acestui tip de ciclu.

Ca urmare a activării lui MEMRD (= „1") sau $\overline{MEMW}$ (= „0") apare semnalul MEMRW care în conjuncție cu semnalul DRSEL, ce indică faptul că se solicită o adresă din cadrul memoriei RAM, poziționează bistabilul RAS pe frontul căzător al ceasului Ø1. Imediat după aceasta apare semnalul $\overline{RAS1}/\overline{RAS2}$, după cum adresa de pe magistrala de adrese a sistemului indică un acces la al doilea sau al treilea bloc de 16 Kocteți al memoriei ($\overline{MBANK_1}$ sau $\overline{MBANK_2}$). În acest moment magistrala de adrese internă a memoriei RAM dinamice conține adresa de rînd, biții $ABUS_0$—$ABUS_6$ ai adresei, deoarece CASE = „0" și nefiind vorba de un ciclu de reîmprospătare RAS-RFE = „0" (fig. 4.20*b*). Adresa de rînd este memorată în circuitele selectate pe frontul negativ al semnalului $\overline{RAS1}/\overline{RAS2}$. Poziționarea lui RAS are ca efect punerea bistabilului CASE (CAS Early) pe frontul crescător al lui Ø2; în acest moment conținutul magistralei interne de adrese a memoriei dinamice devine egal cu adresa de coloană, biții $ABUS_7$ — $ABUS_{13}$ ai adresei (fig. 4.20*b*). Pentru a lăsa un timp de stabilire a adresei de coloană pe magistrala internă, semnalul $\overline{CAS}$ = „0" apare după 50 ns de la poziționarea lui CASE, întîrzierea fiind realizată de circuitul RC plasat la intrarea porții cu intrare de tip trigger Schmitt 7413. Adresa de coloană e memorată în circuitele activate prin $\overline{CAS1}/\overline{CAS2}$ pe frontul negativ al acestui semnal. În acest moment, $\overline{WRITE}$ fiind egal cu „1", ciclul de citire a memoriei este lansat, informația utilă urmînd să apară pe magistrala internă de date $DRDO_i$, $i = 0—7$, la cel mult 165 ns după frontul negativ al lui $\overline{CAS1}/\overline{CAS2}$, ceea ce înseamnă o întîrziere:

$$t = 100 \text{ ns} + 50 \text{ ns} + 50 \text{ ns} + 165 \text{ ns} = 365 \text{ ns}$$

$$(T_{\varnothing 1}) \quad (T_{\varnothing 1 \varnothing 2}) \quad (T_{\varnothing 2-\overline{CAS}}) \quad (T_{\overline{CAS}-date})$$

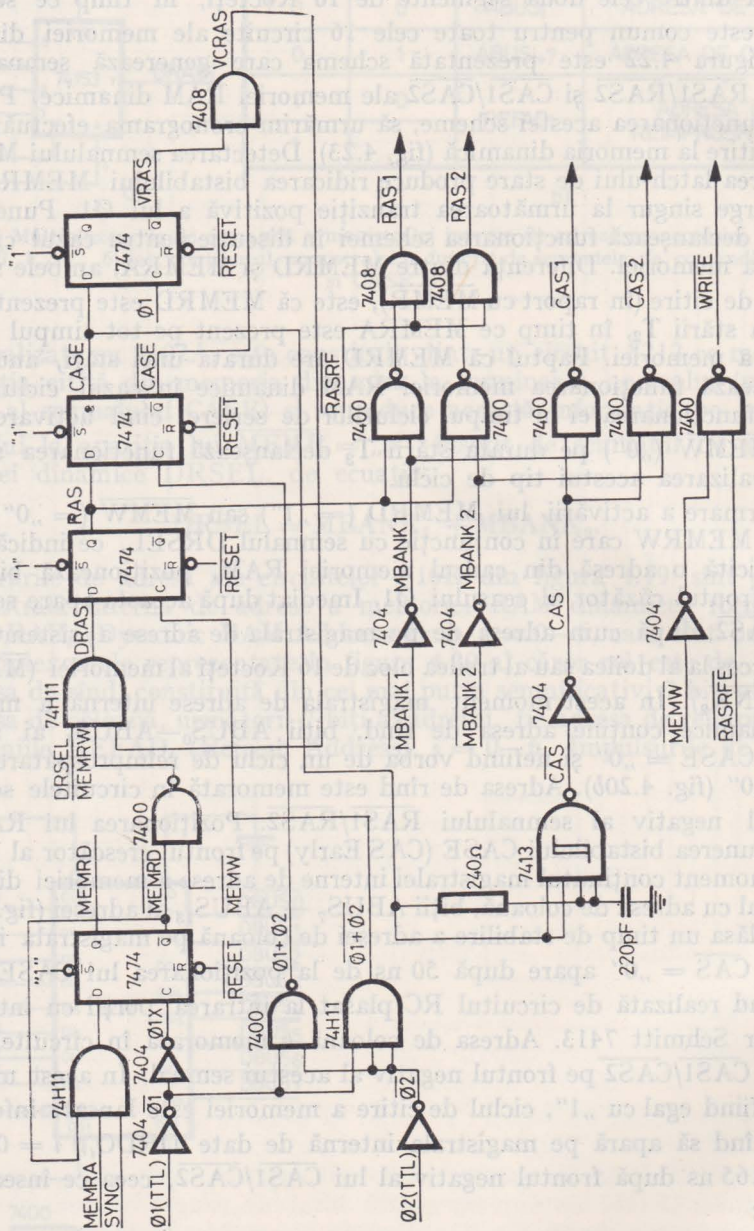


Fig. 4.22. Schema de generare a semnalelor de comandă ale memoriei dinamice

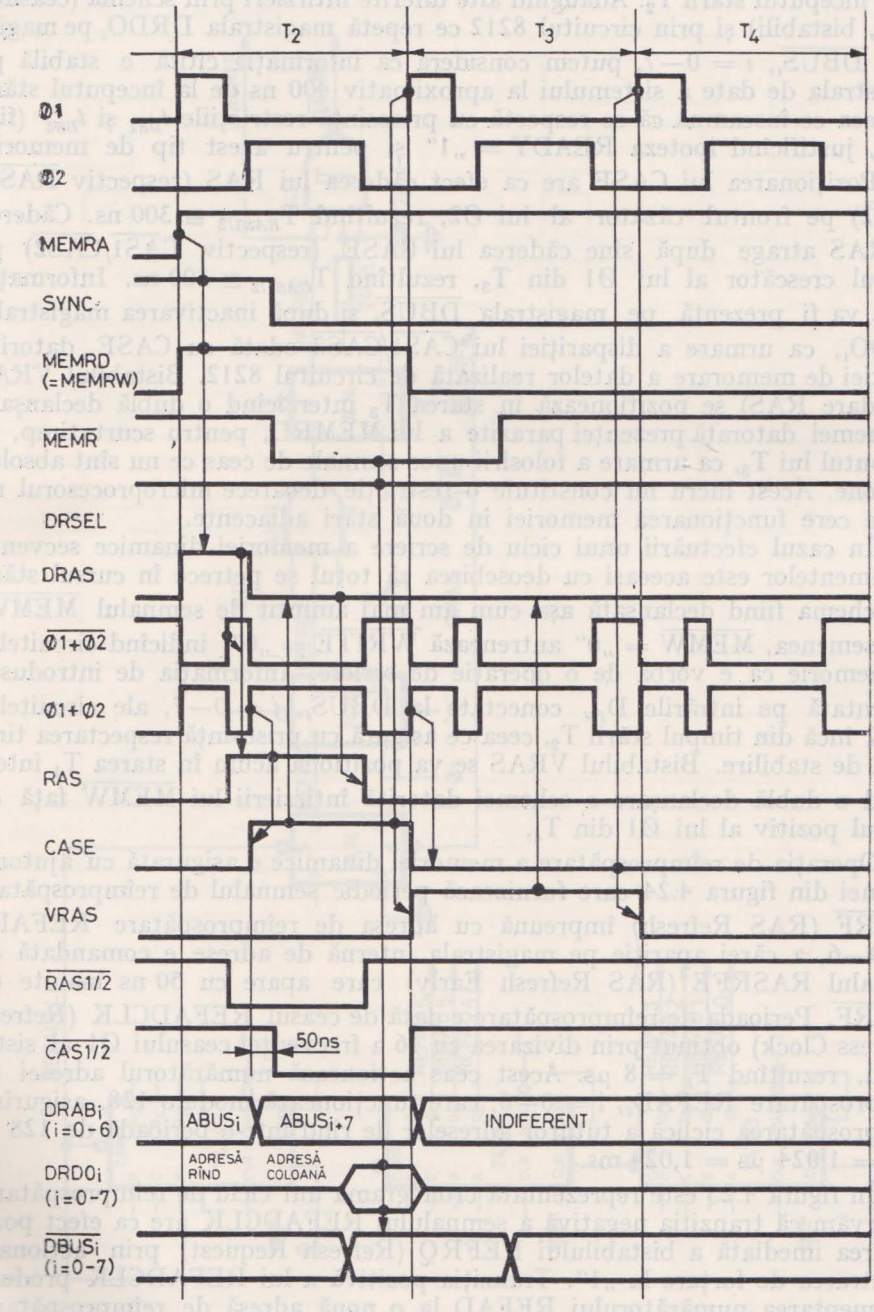


Fig. 4.23. Cronograma unui ciclu de citire a memoriei dinamice

de la începutul stării T_2. Adăugînd alte diferite întîrzieri prin schemă (ceasuri, porți, bistabili) și prin circuitul 8212 ce repetă magistrala $DRDO_i$ pe magistrala $\overline{DBUS}_i$, $i = 0—7$, putem considera că informația citită e stabilă pe magistrala de date a sistemului la aproximativ 400 ns de la începutul stării T_2, ceea ce înseamnă că se respectă cu prisosință restricțiile t_{DS1} și t_{DS2} (fig. 4.15), justificînd ipoteza READY = „1" și pentru acest tip de memorie.

Poziționarea lui CASE are ca efect căderea lui RAS (respectiv $\overline{RAS1}$/$\overline{RAS2}$) pe frontul căzător al lui Ø2, rezultînd $T_{RAS1/2} \cong 300$ ns. Căderea lui RAS atrage după sine căderea lui CASE (respectiv $\overline{CAS1}$/$\overline{CAS2}$) pe frontul crescător al lui Ø1 din T_3, rezultînd $T_{CAS1/2} \cong 300$ ns. Informația citită va fi prezentă pe magistrala $\overline{DBUS}_i$ și după inactivarea magistralei $DRDO_i$, ca urmare a dispariției lui $\overline{CAS1}$/$\overline{CAS2}$ odată cu CASE, datorită funcției de memorare a datelor realizată de circuitul 8212. Bistabilul VRAS (Validare RAS) se poziționează în starea T_3 interzicînd o dublă declanșare a schemei datorată prezenței parazite a lui MEMRD, pentru scurt timp, la începutul lui T_3, ca urmare a folosirii unor semnale de ceas ce nu sînt absolut sincrone. Acest lucru nu constituie o restricție, deoarece microprocesorul nu poate cere funcționarea memoriei în două stări adiacente.

În cazul efectuării unui ciclu de scriere a memoriei dinamice secvența evenimentelor este aceeași cu deosebirea că totul se petrece în cursul stării T_3, schema fiind declanșată așa cum am mai amintit de semnalul $\overline{MEMW}$. De asemenea, $\overline{MEMW}$ = „0" antrenează $\overline{WRITE}$ = „0", indicînd circuitelor de memorie că e vorba de o operație de scriere. Informația de introdus e prezentată pe intrările D_{IN} conectate la $\overline{DBUS}_i$, $i = 0—7$, ale circuitelor F16K încă din timpul stării T_2, ceea ce asigură cu prisosință respectarea timpului de stabilire. Bistabilul VRAS se va poziționa acum în starea T_4 interzicînd o dublă declanșare a schemei datorită întîrzierii lui $\overline{MEMW}$ față de frontul pozitiv al lui Ø1 din T_4.

Operația de reîmprospătare a memoriei dinamice e asigurată cu ajutorul schemei din figura 4.24 care furnizează periodic semnalul de reîmprospătare $\overline{RASRF}$ (RAS Refresh) împreună cu adresa de reîmprospătare $REFAD_i$, i = 0—6, a cărei apariție pe magistrala internă de adrese e comandată de semnalul RASRFE (RAS Refresh Early) care apare cu 50 ns înainte de $\overline{RASRF}$. Perioada de reîmprospătare e dată de ceasul REFADCLK (Refresh Address Clock) obținut prin divizarea cu 16 a frecvenței ceasului Ø1 al sistemului, rezultînd $T_R = 8$ μs. Acest ceas acționează numărătorul adresei de reîmprospătare $REFAD_i$, i = 0—6, care funcționează modulo 128, asigurînd reîmprospătarea ciclică a tuturor adreselor de rînd într-o perioadă de $128 \times 8 = 1\,024$ μs = 1,024 ms.

În figura 4.25 este reprezentată cronograma unui ciclu de reîmprospătare. Observăm că tranziția negativă a semnalului REFADCLK are ca efect poziționarea imediată a bistabilului REFRQ (Refresh Request) prin acționare pe intrarea de forțare la „1". Tranziția pozitivă a lui REFADCLK produce incrementarea numărătorului REFAD la o nouă adresă de reîmprospătare. La începutul stării T_3 valoarea lui REFRQ este trecută în REFRQB (Refresh

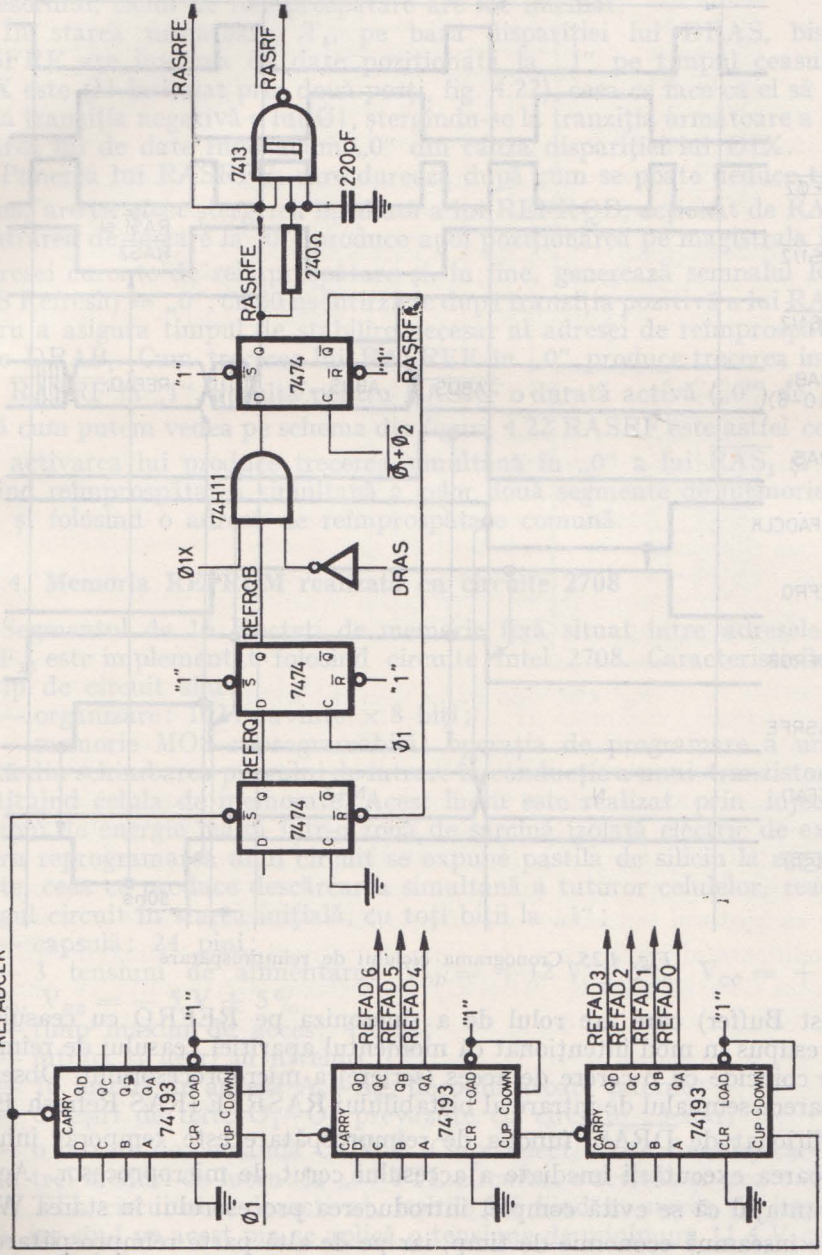


Fig. 4.24. Generarea semnalelor de reîmprospătare

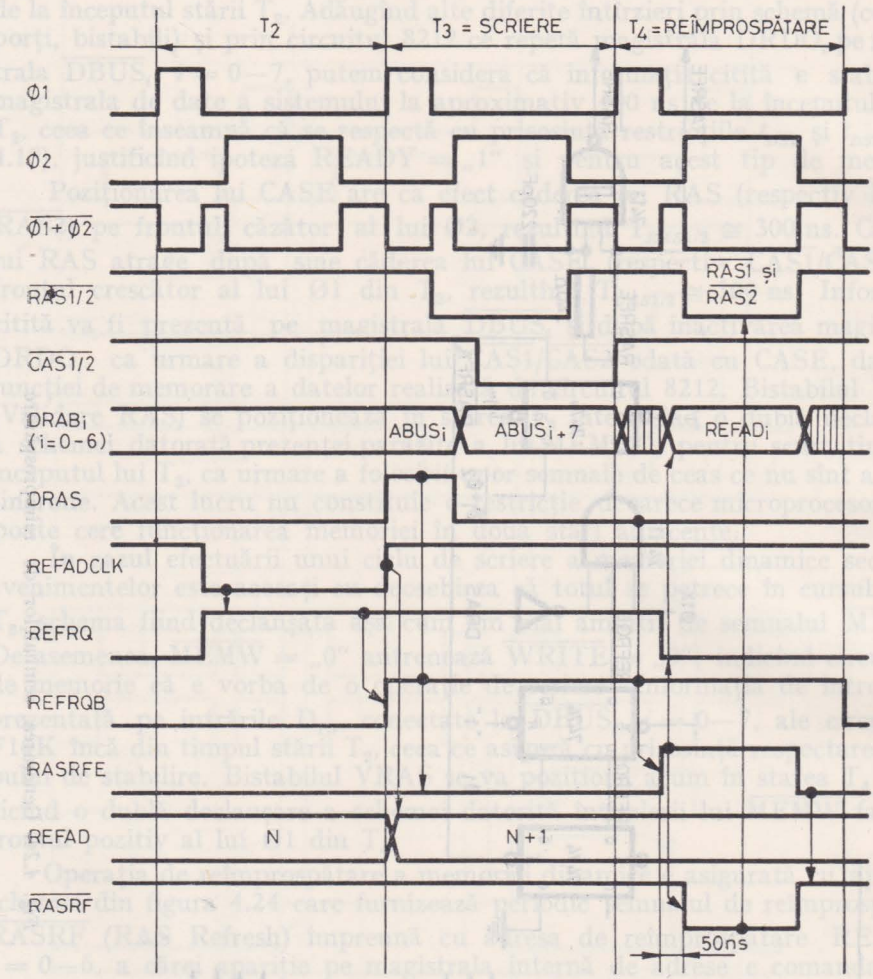


Fig. 4.25. Cronograma ciclului de reîmprospătare

Request Buffer) care are rolul de a sincroniza pe REFRQ cu ceasul Ø1. Am presupus în mod intenționat că momentul apariției ceasului de reîmprospătare coincide cu o cerere de acces (scriere) a microprocesorului. Observăm că deoarece semnalul de intrare al bistabilului RASRFE (RAS Refresh Early) e condiționat de $\overline{\text{DRAS}}$, funcția de reîmprospătare este temporar inhibată în favoarea executării imediate a accesului cerut de microprocesor. Aceasta are avantajul că se evită complet introducerea procesorului în starea WAIT, ceea ce înseamnă economie de timp, iar pe de altă parte reîmprospătarea nu are de suferit, ea fiind totdeauna executată în starea următoare, avînd în vedere că microprocesorul nu poate repeta cererea de acces în două stări

succesive. Desigur că dacă REFRQB nu coincide cu o cerere de acces a microprocesorului, ciclul de reîmprospătare are loc imediat.

În starea următoare, T_4, pe baza dispariției lui DRAS, bistabilul RASFRE are intrarea de date poziționată la „1" pe timpul ceasului Ø1 (Ø1X este Ø1 întîrziat prin două porți, fig. 4.22), ceea ce face ca el să devină „1" la tranziția negativă a lui Ø1, ștergîndu-se la tranziția următoare a lui Ø2, intrarea lui de date fiind acum „0" din cauza dispariției lui Ø1X.

Punerea lui RASRFE, care durează după cum se poate deduce timp de 300 ns, are ca efect ștergerea imediată a lui REFRQB, acționat de $\overline{\text{RASRFE}}$ pe intrarea de forțare la „0", produce apoi poziționarea pe magistrala internă a adresei curente de reîmprospătare și, în fine, generează semnalul $\overline{\text{RASRF}}$ (RAS Refresh) = „0", cu 50 ns întîrziere după tranziția pozitivă a lui RASRFE pentru a asigura timpul de stabilire necesar al adresei de reîmprospătare pe liniile DRAB_i. Cum trecerea lui RASRFE în „0" produce trecerea imediată a lui $\overline{\text{RASRF}}$ în „1", rezultă pentru $\overline{\text{RASRF}}$ o durată activă („0") de 250 ns. După cum putem vedea pe schema din figura 4.22 $\overline{\text{RASRF}}$ este astfel conectat încît activarea lui produce trecerea simultană în „0" a lui $\overline{\text{RAS}_1}$ și $\overline{\text{RAS}_2}$, cauzînd reîmprospătarea simultană a celor două segmente de memorie dinamică și folosind o adresă de reîmprospătare comună.

4.2.3.4. Memoria REPROM realizată cu circuite 2708

Segmentul de 16 Kocteți de memorie fixă situat între adresele C000-FFFF_{16} este implementat folosind circuite Intel 2708. Caracteristicile acestui tip de circuit sînt:

— organizare: 1024 cuvinte × 8 biți;

— memorie MOS reprogramabilă: operația de programare a unui bit constă din schimbarea pragului de intrare în conducție a unui tranzistor MOS constituind celula de memorare. Acest lucru este realizat prin injecția de electroni de energie înaltă într-o zonă de sarcină izolată electric de exterior. Pentru reprogramarea unui circuit se expune pastila de siliciu la raze ultraviolete, ceea ce produce descărcarea simultană a tuturor celulelor, readucînd întregul circuit în starea inițială, cu toți biții la „1";

— capsulă: 24 pini;

— 3 tensiuni de alimentare: $V_{DD} = +12\text{ V} \pm 5\%$, $V_{CC} = +5\text{ V} \pm 5\%$, $V_{BB} = -5\text{ V} \pm 5\%$;

— timp maxim de acces: 450 ns;

— intrări și ieșiri în niveluri TTL;

— 10 intrări de adresă A_0—A_9; decodificatoare interne;

— 8 ieșiri de date O_1—O_8 prevăzute cu circuite „3-stări";

— o intrare de comandă $\overline{\text{CS}}$/WE (Chip Select/Write Enable) care poate selecta trei moduri de lucru: la „0" TTL circuitul e activat și poate fi citit, la „1" TTL circuitul e dezactivat, ieșirile lui fiind în starea de impedanță înaltă, iar cînd pe acest pin se aplică o tensiune de minimum 11,4 V circuitul trece în mod de programare;

— o intrare de comandă PROGRAM pe care se aplică impulsurile de 26 V în timpul operației de programare.

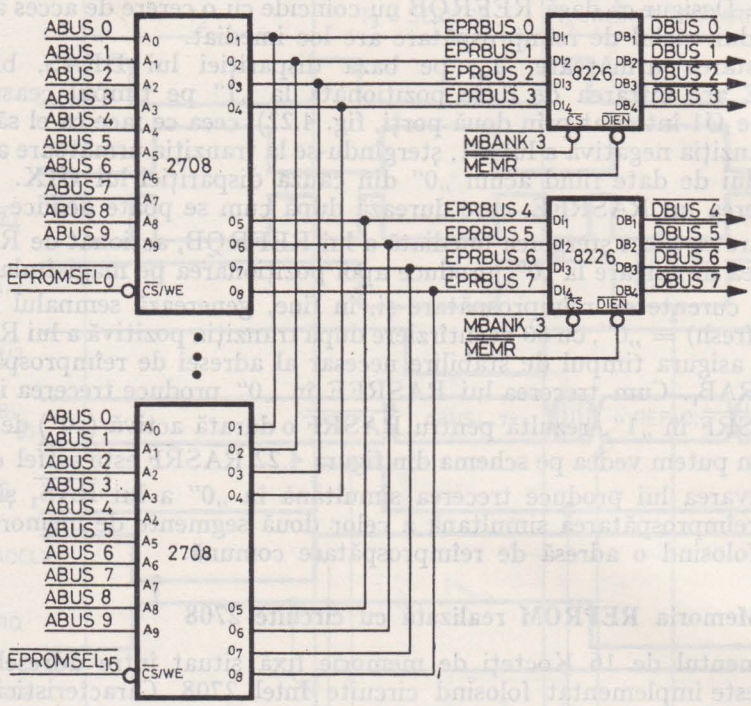


Fig. 4.26. Organizarea memoriei REPROM

Organizarea memoriei REPROM e prezentată în figura 4.26. Intrările de adresă A_0—A_9 ale circuitelor sînt legate direct la liniile magistralei de adrese $ABUS_0$—$ABUS_9$, în timp ce ieșirile O_1—O_8 sînt conectate pe o magistrală internă $EPRBUS_i$, i = 0—7, (EPROM Bus) pe care toate cele 16 circuite 2708 sînt legate în paralel. Activarea unui singur circuit la un moment dat se face cu ajutorul semnalelor de selecție $\overline{EPROMSEL_i}$, i = 0—15, furnizate de decodificatorul din figura 4.14*d*. Cum decodificatorul e validat de $\overline{MBANK_3}$ pe intrarea G1 și de $\overline{MEMR} \cdot \overline{MEMRD}$ pe intrarea G2, rezultă că semnalele $\overline{EPROMSEL_i}$, i = 0—15, devin active („0") în ciclurile de citire a memoriei REPROM la începutul stării T_2 (pe baza lui $\overline{MEMRD}$, același cu cel folosit la memoria dinamică, fig. 4.22 și 4.23) și durează pînă la terminarea lui $\overline{MEMR}$ în urma frontului crescător al lui Ø2 din T_3. Datele de pe magistrala internă sînt reproduse pe magistrala de date a sistemului, pe timpul lui $\overline{MEMR}$, cu ajutorul a două circuite 8226. Acestea din urmă fiind inversoare, rezultă că datele se înscriu în memoria REPROM în forma folosită de procesor. Nu am considerat necesară prezentarea cronogramei accesului la memoria REPROM, deoarece organizarea acestui tip de memorie are elemente comune cu cazurile tratate anterior. Vom remarca totuși că lansarea citirii se face cu un semnal avansat, folosind un circuit de separare între magistrala de date

internă și cea a sistemului, ceea ce permite și în acest caz să se considere READY = „1". În adevăr, timpul de obținere a ieșirii valide este de maximum 450 ns de la stabilirea adresei și de maximum 120 ns de la selectarea circuitului, fapt care înseamnă că datele furnizate pe magistrala sistemului sînt valide cu mult înainte de Ø1 din T_3. Pe de altă parte, pentru a obține performanțele de viteză specificate mai sus este necesar ca semnalul de selecție $\overline{CS}$ să fie menținut la „1" în perioadele de inactivitate a circuitului și adus la „0" numai cu prilejul accesului (cu alte cuvinte $\overline{CS}$ trebuie să fie „pulsat"). Acest deziderat se realizează prin folosirea lui $\overline{MEMRD}$ ca semnal avansat de lansare a citirii, care prezintă față de $\overline{MEMRA}$ avantajul că nu apare decît în stările T_2; în cazul folosirii lui $\overline{MEMRA}$ accesele succesive la același circuit ar avea ca efect $\overline{CS}$ = „0", permanent la circuitul respectiv.

4.2.4. SISTEMUL DE ÎNTRERUPERI

Sistemul de întreruperi al SD-8080 are rolul de a permite sincronizarea activității procesorului cu evenimentele externe care pot surveni la momente de timp aleatoare față de derularea programului din microprocesor.

Funcționarea acestui mecanism de sincronizare presupune atît aspecte de natură hardware, cît și aspecte de natură software, pe care le vom releva în continuare. Condiția preliminară obligatorie pentru intrarea în acțiune a sistemului de întreruperi este ca el să fie activat de către programul procesorului prin executarea unei instrucțiuni EI (Enable Interrupts). Aceasta produce poziționarea bistabilului intern INTE (Interrupt Enable), în timp ce instrucțiunea DI (Disable Interrupts) are un efect contrar. În asemenea condiții, ridicarea liniei de intrare în microprocesor INT (Interrupt) de către un dispozitiv extern va face ca la sfîrșitul instrucțiunii curente microprocesorul să inițieze un ciclu de tip ÎNTRERUPERE (semnalul $\overline{INTA}$ de pe magistrala de comandă) în care o logică externă special afectată acestei funcțiuni va forța pe magistrala de date o instrucțiune specială (pe un octet) de salt la subrutina de tratare a întreruperii (vezi și § 4.2.2). Există opt asemenea instrucțiuni de salt, numite RST (Restart), care au codul binar:

1	1	N	N	N	1	1	1

unde grupul de trei biți NNN codifică una dintre cele opt cauze de întrerupere posibile. O asemenea instrucțiune are exact același efect ca o instrucțiune de salt la subrutină (CALL), saltul fiind efectuat la o adresă egală cu de 8 ori numărul NNN. De exemplu, instrucțiunea cu codul $D7_{16}$, în care grupul NNN are valoarea 010 va executa mai întîi salvarea adresei curente în stivă, la fel ca orice instrucțiune CALL, după care va avea loc saltul la adresa $8 \cdot NNN = 16_{10}$. Luarea în considerare a întreruperii prin generarea ciclului ÎNTRERUPERE de către microprocesor e însoțită în mod automat de ștergerea bistabilului INTE din procesor, astfel că o întrerupere ulterioară nu poate fi luată

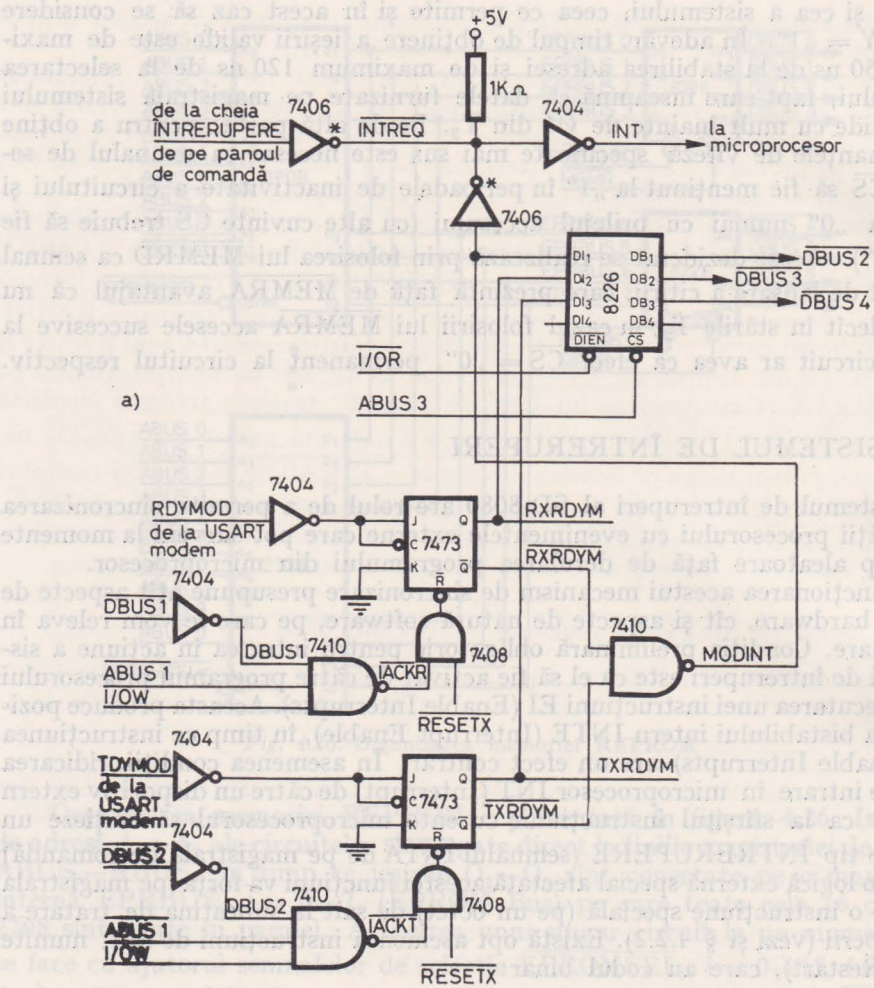


Fig. 4.27. Schema de generare și achitare a întreruperilor (a); logica externă ce inserează instruc-

în considerare decît după terminarea servirii întreruperii curente, cînd subrutina de tratare va reactiva în mod obligatoriu sistemul de întreruperi printr-o instrucțiune EI.

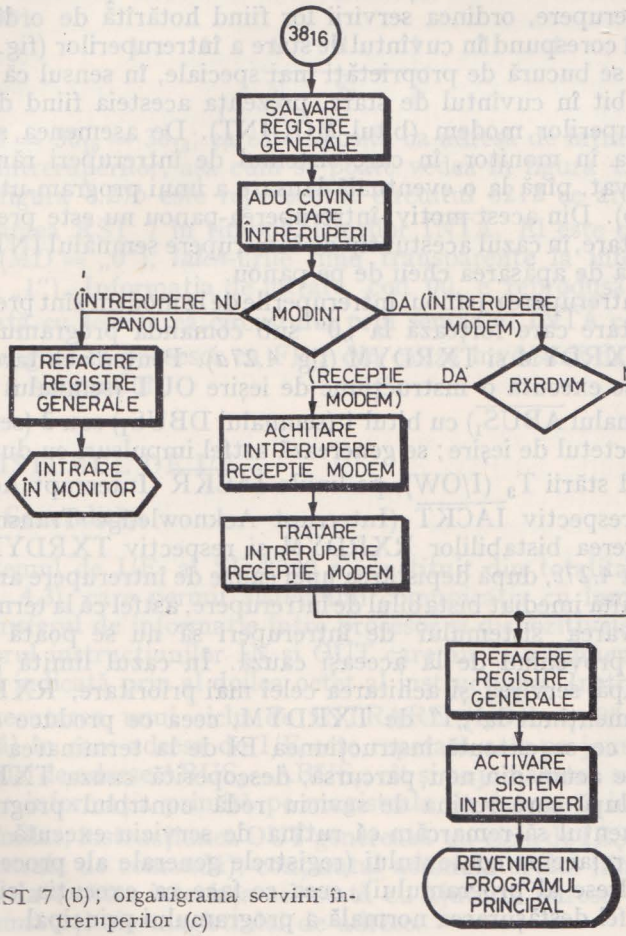
În cazul lui SD-8080 întreruperile pot proveni din trei surse:

— întreruperea-panou, produsă de apăsarea cheii ÎNTRERUPERE de pe panoul de comandă;

— modem gata de emisie, care apare la eliberarea interfeței de modem, atunci cînd aceasta e pregătită să transmită un nou caracter;

— modem gata în recepție, care apare atunci cînd interfața de modem a terminat recepția unui caracter, care este pregătit pentru a fi preluat de către procesor.

După cum se poate vedea în figura 4.27*a* ridicarea liniei INT a microprocesorului este făcută prin activarea liniei externe $\overline{\text{INTREQ}}$ (Interrupt Request),



ția RST 7 (b); organigrama servirii întreruperilor (c)

pe care o folosesc toate cauzele de întrerupere posibile prin conducerea ei la „0" cu circuite cu „colectorul în gol". Întreruperile provenind de la modem sînt ieşirile bistabililor RXRDYM (Receiver Ready Modem) şi TXRDYM (Transmitter Ready Modem) care contribuie la formarea semnalului MODINT (Modem Interrupt), de ecuaţie:

$$\text{MODINT} = \text{RXRDYM} + \text{TXRDYM}.$$

Starea acestor trei semnale poate fi consultată de procesor în mod direct citind *cuvîntul de stare a întreruperilor* aflat pe adresa de intrare $F7_{16}$, MODINT corespunzînd în acest cuvînt de stare bitului 2, iar RXRDYM şi TXRDYM, respectiv biţilor 3 şi 4. Acest lucru este realizat cu ajutorul circuitului 8226 validat în ciclurile de INTRARE cu $\overline{\text{I/OR}}$, atunci cînd adresa de pe magistrală are bitul 3 nul ($ABUS_3$ = „0").

Se observă că între cauzele de întrerupere nu există o prioritate implementată printr-un procedeu hardware. Aşadar la un moment dat pot fi active mai multe cauze de întrerupere, ordinea servirii lor fiind hotărîtă de ordinea de testare a biţilor ce le corespund în cuvîntul de stare a întreruperilor (fig. 4.27*c*). Întreruperea-panou se bucură de proprietăţi mai speciale, în sensul că ei nu-i corespunde nici un bit în cuvîntul de stare, prezenţa acesteia fiind decelată prin absenţa întreruperilor modem (bitul MODINT). De asemenea, servirea ei constă în intrarea în monitor, în care sistemul de întreruperi rămîne în permanenţă dezactivat, pînă la o eventuală lansare a unui program-utilizator prin comanda G (Go). Din acest motiv, întreruperea-panou nu este prevăzută cu un semnal de achitare, în cazul acestui tip de întrerupere semnalul INT avînd o durată determinată de apăsarea cheii de pe panou.

În contrast cu întreruperea-panou, întreruperile de la modem sînt prevăzute cu comenzi de achitare care forţează la „0" sub comanda programului din procesor bistabilii RXRDYM şi TXRDYM (fig. 4.27*a*). Pentru achitarea unei întreruperi-modem se execută o instrucţiune de ieşire OUT (semnalul I/OW) pe adresa FD_{16} (semnalul $\overline{ABUS_1}$) cu bitul 1 (semnalul $DBUS_1$) sau 2 (semnalul $DBUS_2$) activat în octetul de ieşire; se generează astfel impulsuri cu durata de 500 ns, corespunzînd stării T_3 (I/OW), pe liniile $\overline{\text{IACKR}}$ (Interrupt Acknowledge Receiver) şi respectiv $\overline{\text{IACKT}}$ (Interrupt Acknowledge Transmitter), avînd ca efect ştergerea bistabililor RXRDYM şi respectiv TXRDYM. Aşa cum se vede în figura 4.27*c*, după depistarea unei cauze de întrerupere anumite, rutina de serviciu achită imediat bistabilul de întrerupere, astfel că la terminarea serviciului şi reactivarea sistemului de întreruperi să nu se poată genera o nouă întrerupere provenind de la aceeaşi cauză. În cazul limită a două cauze simultane, după servirea şi achitarea celei mai prioritare, RXRDYM, semnalul INT este menţinut la „1" de TXRDYM, ceea ce produce o nouă întrerupere imediat ce se execută instrucţiunea EI de la terminarea rutinei de serviciu, care este acum din nou parcursă, descoperită cauza TXRDYM, achitată şi servită, după care rutina de seviciu redă controlul programului principal. Este momentul să remarcăm că rutina de serviciu execută în mod automat salvarea şi refacerea contextului (registrele generale ale procesorului şi numărătorul de adrese al programului), ceea ce face ca execuţia ei să nu impieteze în nici un fel desfăşurarea normală a programului principal.

Pe de altă parte, bistabilii RXRDYM și TXRDYM din figura 4.27*a* sînt poziționați numai de către fronturile pozitive ale semnalelor de stare ale interfeței modem RDYMOD și TDYMOD (Receiver/Transmitter Ready Modem). Aceasta face ca întreruperea să nu poată fi generată decît o dată în momentul eliberării emițătorului sau la umplerea receptorului interfeței, evitîndu-se o eventuală generare de întreruperi multiple pe tot timpul cît RDYMOD/TDYMOD = „1".

Pentru a completa descrierea sistemului de întreruperi al SD-8080 ne-a mai rămas de precizat faptul că la acceptarea unei întreruperi de către microprocesor, în timpul ciclului ÎNTRERUPERE ($\overline{INTA}$) care apare în acest moment, pe magistrala de date se forțează printr-o logică externă o instrucțiune unică, și anume RST 7, avînd codul FF_{16}. Folosirea unei singure instrucțiuni RST din cele opt disponibile apare foarte normală în contextul existenței unei rutine de serviciu unice care analizează și tratează toate întreruperile posibile. Adresa de salt implicită corespunzînd instrucțiunii RST 7, cod binar:

$$1\ 1\ \underbrace{1\ 1\ 1}_{N\ N\ N}\ 1\ 1\ 1$$

este $8 \cdot 7 = 56_{10} = 38_{16}$; ea este folosită ca adresă de intrare în rutina de serviciu a întreruperilor, așa cum se poate vedea în figura 4.27*c*.

În figura 4.27*b* este reprezentat circuitul 8212 ce are rolul de a insera instrucțiunea RST 7 în timpul ciclurilor $\overline{INTA}$. El este conectat în mod de intrare (MD = „0"), *latch*-urile fiind transparente la informația de intrare (STB = „1"). Informația de intrare, cod 00, e reprodusă pe magistrala de date odată cu selectarea circuitului prin semnalul $\overline{INTA}$ conectat pe intrarea $\overline{DS}_1$, ajungînd în procesor ca FF_{16} din cauza inversării datelor pe magistrala de date a sistemului.

4.2.5. SISTEMUL DE I/E

4.2.5.1. Generalități

Sistemul de I/E al SD-8080 e constituit din totalitatea interfețelor de I/E (fig. 4.3), care permit sistemului comunicația cu lumea exterioară.

Transferul de informație între procesor și dispozitivele periferice se face cu ajutorul instrucțiunilor IN și OUT care au ca argumente o *adresă de I/E* de 8 biți indicată prin al doilea octet al instrucțiunii. Instrucțiunea IN are ca efect executarea unui ciclu de INTRARE ($\overline{I/OR}$ = „0" pe magistrala de comandă) în care adresa de I/E este repetată atît pe jumătatea inferioară a magistralei de adrese $ABUS_0$—$ABUS_7$, cît și pe jumătatea superioară $ABUS_8$—$ABUS_{15}$; informația primită pe magistrala de date este introdusă în acumulator. Similar, instrucțiunea OUT generează un ciclu de IEȘIRE ($\overline{I/OW}$ = „0" pe magistrala de comandă), conținutul acumulatorului fiind trimis pe magistrala de date către interfața selectată cu ajutorul adresei de I/E ce apare pe cele 2 jumătăți ale magistralei de adrese.

Orice implementare practică a unei interfețe de I/E presupune folosirea mai multor adrese:

— o adresă de intrare sau ieșire unde se transferă informația efectivă de I/E; de exemplu în cazul unui display alfanumeric codul ASCII al unui caracter care a fost tastat sau care trebuie afișat pe ecran;

— o adresă de intrare (sau un grup de biți din cadrul octetului de la o adresă de intrare) permițînd microprocesorului să testeze în orice moment starea interfeței sau perifericului respectiv, cum sînt de exemplu semnalele modem gata în recepție (RDYMOD) și modem gata de emisie (TDYMOD) din figura 4.27;

— o adresă de ieșire (sau un grup de biți din cadrul octetului extras pe o adresă de ieșire) permițînd microprocesorului să trimită comenzi către interfața sau perifericul respectiv, cum este de exemplu comanda de alimentare cu o cartelă (EXT.FEED) de la interfața pentru cititorul de cartele.

Există două metode generale de lucru cu un dispozitiv periferic: *în buclă programată și în întreruperi.*

Modul de lucru în buclă programată (*polling*) presupune ca după inițierea unei operații de I/E microprocesorul să rămînă într-o buclă de așteptare programată, testînd unul sau mai mulți biți de stare ai interfeței respective, pînă în momentul cînd sînt reunite toate condițiile necesare pentru realizarea transferului, urmată apoi de schimbul efectiv de informație cu perifericul (vezi § 4.3.2).

Modul de lucru în întreruperi presupune ca după inițierea unei operații de I/E procesorul să-și continue activitatea curentă, urmînd ca la terminarea activității interfața de I/E să semnaleze procesorului acest lucru cu ajutorul unei întreruperi. Acest mod de lucru este în general mai avantajos din punct de vedere al folosirii timpului unității centrale, dar presupune tehnici de programare mai sofisticate pentru sincronizarea între programul principal și rutinele de I/E.

În cadrul lui SD-8080 nu s-a simțit nevoia folosirii modului de lucru în întreruperi decît pentru interfața de modem, care fiind destinată pentru o legătură de comunicație cu un alt calculator își desfășoară activitatea în mod cu totul asincron față de programul din SD-8080, în timp ce restul interfețelor sînt proiectate pentru a funcționa în buclă programată.

În cele ce urmează va fi prezentat numai modul de repartizare a adreselor de I/E în cadrul lui SD-8080, detaliile de proiectare hardware și de programare ale interfețelor fiind prezentate în capitolul 6.

4.2.5.2. Repartizarea adreselor de I/E

Adresa de I/E, fiind exprimată pe 8 biți, permite codificarea a maximum $2^8 = 256$ de adrese de intrare și 256 de adrese de ieșire. Cum folosirea întregului spațiu de adresă nu e în general necesară și decodificarea a 256 de adrese este neeconomică din punct de vedere al implementării, există o variantă simplificată de adresare, așa-zisa „selecție liniară", în care fiecare bit al adresei selectează atunci cînd este activat o anumită interfață. În acest caz logica de

decodificare practic dispare, dar dimensiunea spațiului adreselor de I/E se reduce la 8 combinații, deoarece o adresă validă nu poate cuprinde în situația aceasta decît un singur bit activat.

În cadrul sistemului de I/E al SD-8080 s-au folosit ambele metode. După cum se poate vedea în tabelele 4.3 și 4.4, reprezentînd adresele de intrare, respectiv de ieșire folosite în sistem, cei mai puțin semnificativi 5 biți ai adresei au fost utilizați într-o schemă de selecție liniară, fiecare bit fiind activ în starea „0", deoarece circuitele de I/E folosite au în general intrările de selecție active jos, în timp ce cei mai semnificativi 3 biți au fost complet decodificați. În acest fel s-a obținut un spațiu de 12 adrese de intrare, respectiv 13 adrese de ieșire, în condițiile folosirii unei scheme de decodificare foarte simple.

Tabelul 4.3. **Adresele de intrare folosite în cadrul lui SD-8080**

Binar 7 6 5 4 3 2 1 0	HEX	Semnificație	Observații
1 1 1 1 1 1 1 0	FE	Intrare date consolă	—
1 1 1 1 1 1 0 1	FD	Stare I/E	BIT 0 = TXRDY (ieșire consolă GATA) BIT 1 = RXRDY (intrare consolă GATA) BIT 2 = PCHRDY (Perforator bandă GATA) BIT 3 = RDRDY (Cititor bandă GATA) BIT 4 = CRRDY (Cititor cartele GATA) BIT 5 = DATRDY (Date cit. cartele GATA) BIT 6 = CRBUSY (Cit. cartele OCUPAT) BIT 7 = PRRDY (Imprimantă GATA)
1 1 1 1 1 0 1 1	FB	Intrare date cititor bandă	—
1 1 1 1 0 1 1 1	F7	Viteză de transmisie serie consolă și cuvînt de stare a întreruperilor	BIT 2 = MODINT (ÎNTR. MODEM) BIT 3 = RXDYM (ÎNTR. REC. MODEM) BIT 4 = TXRDYM (ÎNTR. EMISIE MODEM)

BIT	BIT	VIT. (Bd)
1	0	110
0	0	
0	1	200
1	0	300
1	1	600

Tabelul 4.3. (continuare)

Binar 7 6 5 4 3 2 1 0	HEX	Semnificație	Observații
1 1 1 0 1 1 1 1	EF	Intrare date cititor de cartele	Se fac două citiri succesive pe această adresă pentru a primi cele 12 coloane ale codului Hollerith, după cum urmează: OCTETUL 1 — OCTETUL 2 BIT 0=COL.1 — BIT 0=COL.0 BIT 1=COL.2 — BIT 1=COL.11 . — BIT 2=COL.12 . — BIT 3, 4, 5, 6, = „0" BIT 7=COL.8 — BIT. 7=COL. 9
0 0 0 1 1 1 1 1	1F	Nefolosită	—
0 0 1 1 1 1 1 1	3F	Nefolosită	
0 1 0 1 1 1 1 1	5F	Intrare date USART modem	Datele se citesc în formă complementată
0 1 1 1 1 1 1 1	7F	Intrare stare USART modem	Cuvîntul de stare se citește complementat (vezi specificații 8251)
1 0 0 1 1 1 1 1	9F	Intrare date USART casete	Datele se citesc în formă complementată
1 0 1 1 1 1 1 1	BF	Intrare stare USART casete	Cuvîntul de stare se citește complementat (vezi specificații 8251)
1 1 0 1 1 1 1 1	DF	Intrare stare periferic casetă	BIT 0 = ON LINE BIT 1 = WRITE PERMIT BIT 2 = GAP BIT 3 = CASSETTE LOADED BIT 4 = EOTOM BIT 5 = EOTO BIT 6 = BUSY BIT 7 = EOT

Pe aceste tabele se poate observa de asemenea că anumite interfețe, cum ar fi cea a perforatorului rapid de bandă, nu necesită decît o adresă de transfer de informații (adresa de ieșire FB_{16}) și un bit de stare pe o adresă de intrare (bitul 2 pe adresa de intrare FD_{16}), în timp ce alte interfețe necesită atît adrese de intrare-stare, adrese de I/E date, cît și adrese de ieșire-comenzi (cazul interfeței pentru unitățile de casete magnetice). Remarcăm de asemenea că pe o singură adresă de intrare, FD_{16}, au fost regrupați biți de stare provenind de la 5 interfețe diferite, ceea ce economisește spațiul de adresare cu prețul unei simple instrucțiuni de mascare a biților nesemnificativi în fiecare rutină de I/E specifică. Aceeași remarcă se poate face referitor la adresele de ieșire, de exemplu pe adresa FD_{16} sînt reuniți bitul de comandă al cititorului de bandă

Tabelul 4.4. Adresele de ieşire folosite în cadrul lui SD-8080

Binar 7 6 5 4 3 2 1 0	HEX	Semnificaţie	Observaţii
1 1 1 1 1 1 1 0	FE	Ieşire date consolă	—
1 1 1 1 1 1 0 1	FD	Comanda cititor de bandă consolă (TTY) şi achitare întreruperi	BIT 0 = Comandă un pas la cititorul de bandă al TTY (datele apar pe intrarea de date de la consolă) BIT 1 = Achitare într. rec. modem BIT 2 = Achitare într. emisie modem
1 1 1 1 1 0 1 1	FB	Ieşire date perforator bandă	—
1 1 1 1 0 1 1 1	F7	Stabilire viteze de transmisie serie BIŢII 7,6,5 = modem BIŢII 4,3,2, = nefolosiţi BIŢII 1, 0 = consolă	modem: 7 6 5 — COD 0 = 110Bd; COD 1 = 200Bd; COD 2 = 300Bd; COD 3 = 600Bd; COD 4 = 1200Bd nefolosiţi: 4 3 2 consolă: 1 0 — 0 = 110Bd; 1 = 200Bd; 2 = 300Bd; 3 = 600Bd; —
1 1 1 0 1 1 1 1	EF	Comandă alimentarea unei cartele (Ext. Feed)	Octetul de ieşire poate fi oricare
0 0 0 1 1 1 1 1	1F	Nefolosită	—
0 0 1 1 1 1 1 1	3F	Nefolosită	—
0 1 0 1 1 1 1 1	5F	Ieşire comenzi USART modem	Datele se trimit complementate
0 1 1 1 1 1 1 1	7F	Ieşire date USART modem	Cuvîntul de comandă se trimite negat (vezi specificaţii 8251)
1 0 0 1 1 1 1 1	9F	Ieşire date USART casete	Datele se trimit complementate
1 0 1 1 1 1 1 1	BF	Ieşire comenzi USART casete	Cuvîntul de comandă se trimite negat (vezi specificaţii 8251)
1 1 0 1 1 1 1 1	DF	Ieşire comenzi periferic casetă	BIT 0 = FORWARD BIT 1 = RESET BIT 2 = REVERSE BIT 3 = VIT1/VIT2 BIT 4 = WRITE MODE BIT 5 = CAS1 BIT 6 = CAS2 BIT 7 = nefolosit
1 1 1 1 1 1 1 1	FF	Ieşire date imprimantă	BIŢII 0—5 = codul caracterului de tipărit BIT 6 = Indiferent BIT 7 = Comandă tipărire linie; primul caracter al liniei este netipărit şi constituie codul de salt (normal = 1)

al Teletype-ului și biții de achitare ai întreruperilor provenind de la modem. Există de asemenea și situația limită cînd o comandă este implicit dată prin simpla selectare a unei adrese de ieșire, octetul de ieșire neavînd nici o semnificație, cum ar fi cazul comenzii de alimentare (EXT. FEED) a cititorului de cartele.

4.3. RESURSE SOFTWARE

Pentru SD-8080 sînt disponibile trei categorii de programe: monitorul, asamblorul și limbajul BASIC.

a. *Monitorul* este programul care exercită totdeauna comanda funcționării calculatorului la orice inițializare sau punere sub tensiune. El este permanent rezident în memorie, fiind înscris în memorie fixă PROM. Monitorul poate fi privit ca un complement indispensabil al hardware-ului, întrucît doar prin intermediul lui operatorul poate avea acces la registrele unității centrale și la numărătorul de adrese al programului. Execuția oricărui program se lansează numai cu ajutorul monitorului. Tot cu ajutorul său se pot fixa conținuturile registrelor generale sau al indicatorului stivei, se poate scrie sau altera conținutul memoriei, se poate muta o zonă de memorie peste alta, se poate extrage, respectiv introduce, un program din memorie pe un suport extern sau invers și, în fine, se pot modifica vitezele de lucru ale interfețelor-serie. Monitorul poate fi oricînd chemat în execuție prin apăsarea cheii ÎNTRERUPERE de pe panoul de comandă. La intrarea în monitor se salvează în întregime starea unității centrale și se dezactivează sistemul de întreruperi pentru a preîntîmpina orice ieșire nedorită din monitor. Programul în execuție poare fi reluat de la început sau din punctul unde a fost întrerupt folosind comanda G.

b. *Asamblorul* transformă programele scrise în limbaj de asamblare 8080 în formă-obiect direct executabilă pe SD-8080. Programul-sursă este introdus pe bandă perforată sau pe cartele, iar ca rezultat al asamblării se obține *listing*-ul programului-sursă cu liniile numerotate și cu conținutul-memorie corespunzător în dreptul fiecărei instrucțiuni, asociat cu o bandă binară conținînd forma-obiect a programului în format direct executabil pe SD-8080.

c. *Limbajul BASIC* este un limbaj de nivel înalt de tip interpretativ și conversațional la care programele se execută pornind direct de la forma-sursă înmagazinată în memoria SD-8080. Compilarea are loc linie cu linie în timpul execuției, astfel că nu există formă intermediară de traducere a programului; compilarea este simplificată prin absența fazei de generare a codului-mașină. Dezavantajul este că o linie care este executată de mai multe ori în cursul unui program va trebui recompilată de fiecare dată; pe de altă parte, în acest caz timpul de execuție al unui program este mai mare decît în cazul unui compilator care generează cod-obiect, pentru că fazele de compilare și execuție sînt întrepătrunse. Marele avantaj al acestui limbaj îl constituie faptul că permite corectarea ușoară a programelor direct de la consolă, prin adăugarea sau scoaterea liniilor de program-sursă în memorie. Interpretorul BASIC este permanent rezident în memorie, fiind înscris în memoria PROM.

4.3.1. MONITORUL

Monitorul SD-8080 este un program înscris în memoria fixă PROM. El acceptă și execută comenzile utilizatorului introduse de la consola-sistem, care este Teletype-ul. Monitorul permite realizarea următoarelor operații:

— listarea conținutului anumitor zone de memorie, comanda D;

— lansarea în execuție a programelor-utilizator, comanda G;

— vizualizarea și modificarea conținutului memoriei și registrelor, comenzile S și respectiv X;

— introducerea datelor hexazecimale în memorie de la consolă, comanda I;

— extragerea datelor/programelor din memorie pe un suport extern (bandă perforată sau casetă magnetică), comanda P;

— introducerea programelor în memorie pornind de la un suport extern (bandă perforată sau casetă magnetică), comanda L;

— stabilirea și modificarea vitezelor de lucru ale interfețelor-serie (consolă și modem), comanda V;

— stabilirea de „puncte de întrerupere" în programul utilizator.

Lansarea în operație a monitorului are loc în trei situații:

— la punerea sub tensiune a sistemului (se pornește de la adresa 0000H care este adresa de început a monitorului);

— la acționarea cheii INIȚIALIZARE de pe panoul de comandă (același lucru ca la punerea sub tensiune);

— la acționarea cheii ÎNTRERUPERE de pe panoul de comandă; în acest caz se lansează monitorul de la adresa 0014H unde se găsește secvența de salvare a stării unității centrale.

La fiecare intrare în monitor se tipărește mesajul de identificare *SD-8080* după care este editat un punct la începutul liniei următoare invitînd utilizatorul să introducă o comandă. Fiecare comandă a monitorului este identificată printr-o literă. Anumite comenzi necesită argumente care sînt, în general, numere hexazecimale formate din cifrele 0 la 9 și literele A la F. Un asemenea număr poate avea de la 1 pînă la 4 cifre hexazecimale, iar numerele mai mari vor fi evaluate modulo 2^{16} (se reține numai ultimele 4 cifre).

În cursul fazei de punere la punct a programelor este necesar să se poată examina evoluția procesului de calcul la diferite momente de timp. Din acest motiv, monitorul posedă o secvență de salvare a stării programului-utilizator care este automat executată la intrarea în monitor cauzată de apăsarea cheii ÎNTRERUPERE de pe panoul de comandă. Mecanismul de salvare a stării poate fi declanșat și în mod programat prin stabilirea unui „punct de întrerupere" (breakpoint) în programul-utilizator cu ajutorul instrucțiunii RST 7 (cod hexazecimal FF).

Exemplu de folosire a monitorului. Presupunem un program încărcat de la 1000H la 1280H; starea sa trebuie examinată în cursul execuției instruc-

ţiunilor de la adresele 1105H şi 1245H. Secvenţa de comenzi la consolă este următoarea:

SD-8080	Pornirea sistemului
.XS 0002-BDD3	Pointerul stivei programului utilizator are o valoare aleatoare la pornirea calculatorului şi trebuie iniţializat pentru a putea executa o comandă G ulterioară
.L	Încărcarea programului utilizator de pe bandă
DV=T	perforată
.S1105 CD-FF	Stabilirea „punctului de întrerupere"
.G1000	Lansarea programului-utilizator
SD-8080	Revenirea în monitor datorită „punctului de întrerupere"
.X	Examinarea stării unităţii centrale

A=01 B=55 C=00 D=00 E=04 F=56 H=14 L=20 M=1420 P=
=1106 S=BDCF

.S1400 03-5	Examinarea şi modificarea unui indicator al programului aflat în memorie (de exemplu un contor)
.S1105 FF-CD	Refacerea adresei alterate anterior
.S1245 3E-FF	Stabilirea noului „punct de întrerupere"
.G1105	Relansarea programului utilizator
SD-8080	Revenirea în monitor datorită „punctului de întrerupere"
.X	Examinarea stării unităţii centrale

A=62 B=87 C=00 D=00 E=87 F=57 H=14 L=80 M=1480 P=1255
S=BDD3

.S1245 FF-3E	Refacerea adresei alterate anterior

4.3.2. SISTEMUL RUTINELOR DE I/E

Sistemul rutinelor de I/E este o colecţie de rutine care pun în funcţiune perifericele. Fiecare periferic al SD-8080 are asociată o asemenea rutină pe care o vom denumi de asemenea şi *driver* (sau *driver* software). Acesta completează posibilităţile de bază ale perifericului făcîndu-l exploatabil prin simpla chemare a rutinei corespunzătoare.

Principiul de bază folosit pentru exploatarea perifericelor în SD-8080 este programarea în buclă (*polling*); aceasta înseamnă că operaţia de I/E nu începe pînă ce perifericul nu este pregătit pentru a putea transfera un nou caracter, timpul de aşteptare fiind petrecut de calculator într-o buclă programată în care se testează unul sau mai mulţi biţi de stare ai perifericului respectiv (fig. 4.28 şi fig. 4.29). Această abordare este comună pentru toate rutinele de I/E, în afară de modem pentru care s-a prevăzut lucrul în întreruperi;

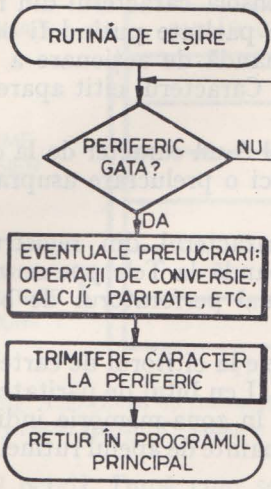


Fig. 4.28. Organigrama generală a unei rutine de ieșire

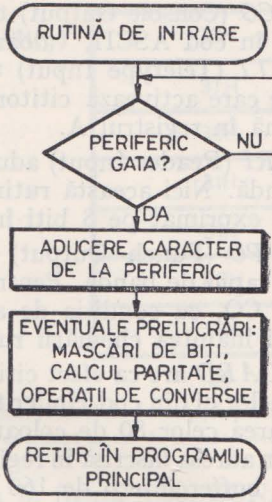


Fig. 4.29. Organigrama generală a unei rutine de intrare

deoarece eventuala comunicație pe linie telefonică cu un alt calculator este un proces cu totul asincron față de prelucrările din SD-8080. Anumite rutine de I/E efectuează și diferite prelucrări specifice pentru perifericul deservit, cum ar fi cazul *driver*-ilor de casetă care implementează prin program unele funcții interne ale unității de casetă sau al *driver*-ilor pentru imprimantă și cititor de cartele, care fac operații de conversie de cod. Caracteristica lor comună este că ele simplifică și unifică operația de programare a I/E, de aceea s-a simțit nevoia introducerii lor în memoria fixă în continuarea monitorului, (vezi Harta memoriei SD-8080, fig. 4.11), astfel încît atît monitorul cît și interpretorul BASIC să poată face apel la ele.

Rutinele de I/E folosesc, de asemenea, o zonă de memorie RAM, necesară pentru diferite buffere și pointeri utilizați în cadrul acestor rutine (vezi Harta memoriei SD-8080, fig. 4.11).

Cu excepția *driver*-ului cititorului de cartele, toate rutinele de I/E funcționează caracter cu caracter, chiar dacă perifericele aferente au un mod de lucru diferit (vezi de exemplu imprimanta sau unitățile de casetă magnetică). Ca regulă generală, rutinele de intrare aduc caracterul în registrul acumulator A, iar rutinele de ieșire își iau caracterul din registrul C. Unele dintre ele folosesc și alte registre în afară de A și C; de aceea este necesară salvarea registrelor alterate de rutină înainte de apelul ei, în cazul cînd valoarea acestora are semnificație pentru programul apelant.

Rutina CI (Console Input) aduce un caracter de la consolă în registrul A. Nu se face nici un control asupra bitului de paritate (cel mai semnificativ bit al caracterului). Codul de transmitere a informației folosit pe această interfață este codul ASCII.

Rutina CO (Console Output) tipărește la consolă caracterul din registrul C, exprimat în cod ASCII, valoarea bitului de paritate putînd fi oarecare.

Rutina TI (Teletype Input) trimite o comandă de acționare a releului din Teletype care activează cititorul de bandă. Caracterul citit apare la ieșirea din rutină în registrul A.

Rutina RI (Reader Input) aduce în registrul A un caracter de la cititorul rapid de bandă. Nici această rutină nu face nici o prelucrare asupra caracterului citit, exprimat pe 8 biți în cod ASCII.

Rutina PO (Punch Output) perforează caracterul din registrul C la perforatorul rapid de bandă. Pentru a perfora bandă la Teletype se va putea folosi rutina CO, cu condiția de a activa manual perforatorul de bandă al Teletype-ului înaintea chemării rutinei.

Rutina CARD are ca efect citirea unei cartele la cititorul de cartele, conversia coloanelor ei din cod Hollerith în cod ASCII cu bitul de paritate egal cu 0 și depozitarea celor 80 de coloane convertite în zona-memorie indicată de utilizator prin adresa înscrisă în registrele H și L înainte de apelul rutinei CARD. Folosește un buffer intern de 160 de caractere.

Rutina PRINT trimite la imprimantă caracterul ASCII primit în registrul C. În prealabil caracterul este convertit în cod EBCDIC. Caracterul CR (Carriage Return) produce tipărirea liniei curente memorate în bufferul intern al imprimantei și saltul la linia următoare, iar caracterul LF (Line Feed) este filtrat. Caracterele care nu fac parte din setul imprimantei sînt înlocuite cu blanc de către rutina de conversie de cod.

Înainte de a descrie *driver*-ii unităților de casete magnetice să facem o scurtă prezentare a modului de organizare a informației pe caseta magnetică. Pe acest suport informația este organizată în fișiere identificate cu ajutorul unui număr format din patru cifre zecimale. Fiecare fișier este alcătuit din unul sau mai multe blocuri conținînd 256 de octeți sau caractere și o informație de control pe doi octeți. Primul este blocul de identificare al fișierului și conține numai 6 octeți: 4 reprezentînd numele fișierului și doi de control. Dacă un fișier are un număr de caractere care nu e multiplu de 256, i se adaugă un număr corespunzător de caractere nule în ultimul bloc. Deși informația este altfel organizată pe suportul fizic, *driver*-ii unităților de casete magnetice transferă informația la și de la programul apelant în mod caracter cu caracter, pentru a păstra unitatea logică a întregului sistem de I/E și compatibilitatea cu rutinele de tratare a suportului-bandă perforată. Există 6 rutine care permit exploatarea casetelor magnetice și anume:

Rutina OPEN este chemată întotdeauna înainte de a începe scrierea unui nou fișier. Ea execută următoarele funcțiuni:

— inițializează unitatea de casetă specificată;

— verifică dacă nu cumva caseta este protejată la scriere;

— caută spațiu liber pe casetă și scrie blocul de identificare a fișierului (*header*).

Înainte de apel utilizatorul trebuie să completeze o zonă de memorie pentru transferul argumentelor către rutinele de casetă numită ZONAF, aceeași pentru toate rutinele de casetă, implantată în memoria RAM și avînd o lungi-

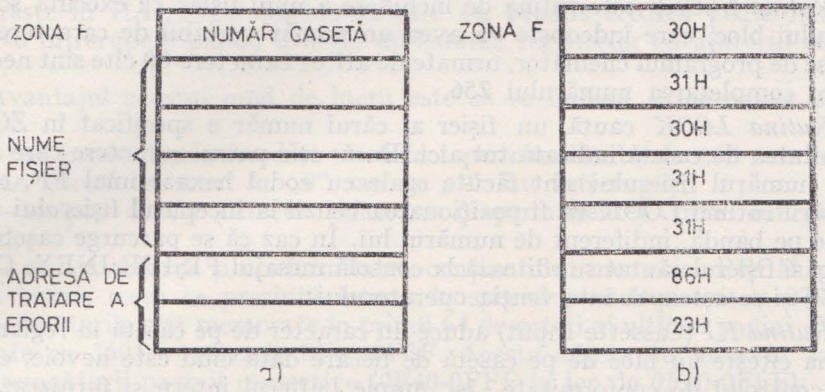


Fig. 4.30. Zona de transmitere a argumentelor rutinelor de casetă (ZONAF): a — formatul general; b — exemplu: caseta 1, număr fișier = 1011 adresa de tratare a erorii 2386H.

me de 7 octeți. După cum se poate vedea în figura 4.30, ZONAF cuprinde următoarele informații:

— numărul casetei cu care se lucrează (1 octet în cod ASCII):

— 0 (cod 30H) pentru caseta 1;

— 1 (cod 31H) pentru caseta 2;

— numele fișierului, 4 octeți reprezentînd codurile ASCII a 4 cifre zecimale formînd numărul (numele) fișierului;

— adresa de tratare a erorii, 2 octeți reprezentînd primul partea joasă și al doilea partea înaltă a adresei unde se dă controlul în cazul cînd se produce o eroare în lucrul cu caseta.

Rutina KO (Cassette Output) primește prin intermediul registrului C caracterul care trebuie scris pe casetă, îl introduce în bufferul propriu, iar în cazul în care bufferul este plin (256 octeți) calculează informația de control (2 octeți) și scrie blocul pe casetă. În timpul scrierii se folosește ca metodă de control citirea după scriere. În caz că se depistează o eroare se dă banda înapoi și se repetă scrierea întregului bloc. Se fac pînă la 10 încercări după care se consideră că suportul este defect și se editează la consolă mesajul ER. SCR. CAS. 1 (sau 2) așteptîndu-se intervenția operatorului. După schimbarea casetei, operatorul va tasta un caracter oarecare la consolă semnalînd *driver*-ului terminarea intervenției sale. Din acest moment se iese din rutina KO și se dă automat controlul secvenței de tratare a erorii indicată prin adresa conținută de ultimii doi octeți din ZONAF.

În cazul în care în cursul scrierii unui fișier se constată că s-a terminat caseta, se editează mesajul de eroare PLIN CAS. 2 (sau 1) și se așteaptă intervenția operatorului (schimbarea casetei) urmată de introducerea unui caracter oarecare la consolă. Înainte de intervenția operatorului *driver*-ul șterge porțiunea deja înscrisă, astfel că pe o casetă nu se pot găsi fișiere incomplete care să poată crea confuzii.

Dacă se încearcă scrierea pe o casetă protejată se editează imediat mesajul de eroare PROT. CAS. 1 (sau 2), fără a se altera conținutul casetei, care va trebui în continuare înlocuită.

Rutina CLOSE este rutina de închidere a unui fișier ce execută scrierea ultimului bloc, care îndeobște va avea un număr variabil de caractere utile trimise de programul chemător, urmate de atîtea caractere 00 cîte sînt necesare pentru completarea numărului 256.

Rutina LOOK caută un fișier al cărui număr e specificat în ZONAF pe unitatea de casetă indicată tot aici. Dacă cele patru caractere care reprezintă numărul fișierului sînt făcute egale cu codul hexazecimal FF, efectul chemării rutinei LOOK va fi poziționarea benzii la începutul fișierului următor de pe bandă, indiferent de numărul lui. În caz că se parcurge caseta fără a se găsi fișierul căutat se editează la consolă mesajul FIȘIER INEX. CAS. 1 (sau 2) și se așteaptă intervenția operatorului.

Rutina KI (Cassette Input) aduce un caracter de pe casetă în registrul A. Rutina citește un bloc de pe casetă de fiecare dată cînd este nevoie, efectuează controlul de conformitate, își umple bufferul intern și furnizează utilizatorului cîte un caracter la fiecare apelare. În cazul în care controlul de conformitate detectează o eroare se încearcă din nou citirea blocului efectuînd iarăși controlul. Dacă după 10 încercări nu se reușește nici o citire corectă se editează mesajul de eroare ER. CIT. CAS. 1 (sau 2) și se așteaptă intervenția operatorului.

Rutina CLEAR șterge în întregime caseta montată pe unitatea al cărei număr este indicat în ZONAF.

4.3.3. PROGRAMELE DE TEST ALE MEMORIEI

Pentru a putea controla în orice moment starea de funcționare a memoriei, SD-8080 este prevăzut cu două programe de test permanent rezidente în PROM. Ele pot fi lansate în execuție cu ajutorul monitorului, presupunînd calculatorul funcțional pînă la acest nivel.

Corespunzător celor două tipuri de memorie folosite în SD-8080 există două programe de test:

— PTROM, care testează memoria PROM;

— PTRAM, care testează memoria RAM.

Programul PTROM. Urmărește testarea memoriei PROM realizate cu două tipuri de circuite: 3601 (organizare 256 cuvinte × 4 biți) și 2708 (organizare 1024 cuvinte × 8 biți). Pentru aceasta memoria e împărțită în pagini corespunzînd distribuției ei fizice pe circuite, după cum urmează:

— adresele 0000—OFFF, memorie realizată cu circuite 3601; o pagină are 256 de octeți și sînt 16 asemenea pagini;

— adresele C000-FFFF, memorie realizată cu circuite 2708; o pagină are 1024 de octeți și sînt 16 asemenea pagini.

PTROM calculează „semnătura" fiecărei pagini ca fiind restul împărțirii lanțului binar format din concatenarea octeților succesivi ai paginii respective la un polinom generator de gradul 16 și o compară cu una din cele 32 de semnături de 16 biți corecte anterior calculate. În caz de eroare programul

se oprește în HALT, iar pagina în care s-a produs eroarea poate fi aflată printr-o întrerupere panou urmată de listarea registrelor cu ajutorul comenzii X.

Avantajul acestui mod de lucru este că se îmbină următoarele posibilități:

— se poate decela imediat un defect permanent: bit la „0" sau la „1", linie de adresă în scurt la „0" sau la „1" sau întreruptă;

— se pot decela defecte aleatoare (nepermanente) prin funcționare buclată;

— se controlează și conformitatea conținutului memoriei PROM cu cel inițial înscris (ceea ce prezintă interes în special pentru circuitele 2708).

Semnăturile sînt memorate în primii 64 de octeți ai ultimei pagini PROM realizate cu 3601 — adresele 0F00-0F3F. Aceste adrese nu intervin la calculul semnăturii paginii respective (0F40-0FFF în loc de 0F00-0FFF). Octetul cel mai puțin semnificativ al semnăturii are adresa mai mică, iar octetul mai semnificativ are adresa următoare.

În timpul execuției, acest program nu folosește de loc memoria RAM: nu se memorează rezultate intermediare sau adrese, nu se fac apeluri de subrutine și nici nu se folosesc instrucțiuni care lucrează cu stiva.

PTROM funcționează în buclă infinită; oprirea se face numai în caz de eroare pe HALT. Pentru a semnala utilizatorului realizarea unui test complet se bate un caracter inefectiv (cod 00) la consolă. În cazul unui test de durată suprimarea tipăririi se va face oprind consola.

Programul PTRAM. Realizează testarea memoriei RAM situate în spațiul de adresă 1000-BFFF. Efectuează o serie de teste succesive după care trimite un caracter inefectiv (cod 00) la consolă și buclează pentru a permite executarea unui test de durată. La fel ca și PTROM nu folosește deloc apeluri de subrutine sau rezultate intermediare în RAM. În caz de eroare programul se oprește în HALT; codul apărut în registrul de adrese de pe panou în urma unei întreruperi, calculatorul fiind operat în modul pas cu pas, indică tipul testului nereușit, iar adresa și tipul erorii pot fi aflate listînd registrele cu comanda X.

PTRAM efectuează următoarele teste:

— „Galoping One" care constă din înscrierea unui cod format din 7 zerouri și o singură unitate ce ocupă succesiv toate pozițiile octetului; acest test se efectuează pentru toate locațiile memoriei, după care toată memoria rămîne la 00;

— se testează dacă toate locațiile din memorie conțin 00; după fiecare citire în locația respectivă se înscrie partea joasă a adresei ei;

— se testează dacă fiecare locație din memorie conține partea joasă a adresei ei; în urma citirii se înlocuiește adresa cu 00;

— se testează interferența liniilor de adresă înscriind codul hexazecimal FF în locația curentă și apoi verificînd că locațiile a căror adresă diferă doar printr-un singur bit au rămas 00; se reface apoi 00 în adresa curentă;

— se înscrie codul 00, FF, 00, FF, ... în toată memoria RAM, apoi se verifică (șah); se reia testul înscriind acum FF, 00, FF, ...;

— se copiază memoria PROM în RAM și se verifică.

4.3.4. LIMBAJUL BASIC

Cum s-a mai amintit, SD-8080 poate fi programat în limbajul conversațional de nivel înalt BASIC (*Beginners' All-purpose Symbolic Instruction Code*). Pentru executarea programului utilizator calculatorul nu mai transformă în acest caz programul-sursă în instrucțiuni mașină care realizează acțiunea descrisă, ci pornește execuția direct de la textul-sursă aflat în memoria principală pe care îl „interpretează" instrucțiune cu instrucțiune. De aceea tipul de compilator cu care este dotat SD-8080 se numește interpretor.

Caracterul conversațional e dat de faptul că activitatea de programare este continuu dirijată de operator prin intermediul consolei, reacția sistemului fiind imediată.

Interpretorul BASIC poate funcționa în două regimuri: de comandă și de execuție.

În *regim de comandă* se pot introduce de la consolă comenzi sau instrucțiuni de program. Comenzile se execută pe loc și permit programatorului realizarea unor funcțiuni auxiliare legate de lansarea în execuție și de întreținerea programelor, după cum urmează:

— comanda LIST produce listarea programului existent în memorie în ordinea crescătoare a numerelor de linie;

— comanda SCRATCH (SCR) este utilizată pentru ștergerea programului existent în memorie; se poate apoi introduce un nou program, toată memoria-utilizator fiind disponibilă;

— comanda RUN este folosită pentru lansarea în execuție a programului-utilizator aflat în memorie;

— comanda SAVE permite obținerea unei copii a programului-sursă aflat în memorie pe un suport extern: casetă magnetică sau bandă perforată;

— comanda LOAD este folosită pentru încărcarea în memorie a unui program-sursă copiat pe un suport extern cu ajutorul comenzii SAVE.

În *regim de comandă* orice linie introdusă de la consolă care începe cu un caracter numeric este considerată instrucțiune și memorată. Instrucțiunile sînt identificate prin etichete constituite din numere întregi cuprinse între 0 și 32 767. Etichetele determină ordinea de execuție a instrucțiunilor și sînt utilizate, de asemenea, în instrucțiunile de salt, pentru referirea punctului unde se transferă controlul. În timpul introducerii programului există următoarele posibilități de corectare a textului-sursă;

— ștergerea ultimului caracter introdus se efectuează acționînd tasta RUBOUT;

— ștergerea unei linii deja introduse se realizează prin tastarea etichetei ei urmate de RETURN.

În *regim de execuție* (introdus prin comanda RUN) are loc execuția programului aflat în memorie care începe cu instrucțiunea cu cel mai mic număr de linie și continuă în ordinea crescătoare a numerelor de linie, indiferent de ordinea introducerii lor în memorie, pînă la:

— depistarea unei erori în program;

— execuția unei instrucțiuni STOP sau END;

— întreruperea de la consolă acționînd tasta BREAK.

Oricare dintre cauzele anterioare produce trecerea automată în *regim de comandă*, ceea ce permite efectuarea de eventuale adăugiri sau corecturi în program, după care el poate fi executat din nou.

Nu intenționăm să facem aici o prezentare completă a posibilităților limbajului BASIC, întrucît considerăm că acest lucru nu face obiectul volumului de față. Tot ce vom spune este că limbajul BASIC cu care este dotat SD-8080 satisface cerințele minime ale limbajului creat la Dartmouth College. Cititorul care dorește să aprofundeze această problemă este sfătuit să consulte referința bibliografică [18].

Pentru ca cititorul să-și poată face totuși o idee asupra caracteristicilor acestui limbaj asemănător cu FORTRAN, dar despre care mulți cred că e mai simplu și mai ușor de învățat, în figura 4.31 este dat un exemplu de folosire a lui. Secvența începe cu comanda GC040 a monitorului care permite lansarea în execuție a interpretorului BASIC. Intrarea acestuia în acțiune este semnalată prin apariția mesajului READY care semnifică faptul că interpretorul e gata să accepte introducerea de instrucțiuni sau comenzi. Programatorul a introdus apoi comanda LOAD specificînd prin litera R (Reader), introdusă ca răspuns la întrebarea „DV =" a sistemului, ca încărcarea să se facă prin intermediul cititorului rapid de bandă. După terminarea încărcării, interpretorul trimite mesajul READY în așteptarea comenzii următoare. Aceasta nu e recunoscută de sistem și e rejectată prin răspunsul de invalidare WHAT?. Comanda LIST care urmează produce listarea programului-sursă anterior încărcat. Se observă că instrucțiunile-sursă sînt de regulă etichetate în progresie aritmetică crescătoare cu rația 10 ceea ce permite eventuala inserare ulterioară de instrucțiuni noi în programul deja existent (vezi liniile 105 și 145). Instrucțiunile 10 la 120 permit tipărirea în linia următoare a textului cu rol explicativ aflat între ghilimele; dacă textul între ghilimele lipsește se efectuează numai saltul la linia următoare (liniile 100 și 110). În linia 130 începe o buclă (asemănătoare cu instrucțiunea DO din FORTRAN) al cărei contor (valoarea rezistenței) înregistrează valori cuprinse între 50 și 1 000 cu pas 50. Linia 140 produce tipărirea pe o linie a valorii rezistenței și a timpului de întîrziere Δt calculat cu formula:

$$\Delta t = (R + 130) \cdot C \cdot \ln \frac{V_{cc} - R \cdot I_{IL}}{V_{cc} - V_{T+}}.$$

Instrucțiunea din linia 145 indică interpretorului sfîrșitul instrucțiunilor din cadrul buclei inițiate în linia 130. Instrucțiunile din corpul buclei sînt executate pentru toate valorile contorului cuprinse între 50 și 1 000 obținute prin incrementarea cu 50 a valorii anterioare. Programul se termină cu instrucțiunea END aflată în linia 150. Lansarea în execuție a acestui program a fost făcută prin intermediul comenzii RUN. După terminarea execuției s-a reintrat în *regim de comandă*, sistemul trimițînd mesajul READY.

```
*SD-8080*
.XS BDC1-
.GC040

READY
LØAD

DV=R

READY
RYT
WHAT?

READY
LIST

10PRINT"CALCULUL UNUI CIRCUIT DE INTIRZIERE R-C CUPLAT"
20PRINT"LA INTRAREA UNUI TRIGGER SCHMITT SN7413"
30PRINT"FØRMULA DE CALCUL ESTE :"
40PRINT"DT = (R+130)*C*LØG(VCC-R*IIL)/(VCC-VT+)"
50PRINT"UNDE :"
60PRINT"R(ØHMI),C(NF),DT(NS),V(V),I(A)"
70PRINT"IIL=1 MA, CURENTUL DE INTRARE AL PØRTII IN 0 LØGIC"
80PRINT"FØRMULA E VALABILA PENTRU TRANZITIA 0-1 LA INTRARE"
90PRINT"PENTRU TRANZITIA 1-0 LA INTRARE SE ELIMINA CØNSTANTA 130"
100PRINT
105PRINT"CAPACITATEA C=220 PF=CØNST."
110PRINT
120PRINT"REZISTENTA (ØHMI)"TAB(30)"INTIRZIEREA (NS)"
130FØRR=50T01000STEP50
140PRINTR,,(R+130)*.22*LØG((5-R*1E-3)/3.3)
145NEXTR
150END

READY
RUN

CALCULUL UNUI CIRCUIT DE INTIRZIERE R-C CUPLAT
LA INTRAREA UNUI TRIGGER SCHMITT SN7413
FØRMULA DE CALCUL ESTE :
DT = (R+130)*C*LØG(VCC-R*IIL)/(VCC-VT+)
UNDE :
R(ØHMI),C(NF),DT(NS),V(V),I(A)
IIL=1 MA, CURENTUL DE INTRARE AL PØRTII IN 0 LØGIC
FØRMULA E VALABILA PENTRU TRANZITIA 0-1 LA INTRARE
PENTRU TRANZITIA 1-0 LA INTRARE SE ELIMINA CØNSTANTA 130

CAPACITATEA C=220 PF=CØNST.

REZISTENTA (ØHMI)          INTIRZIEREA (NS)
 5.00000E 01                1.60564E 01
 1.00000E 02                2.00028E 01
 1.50000E 02                2.37194E 01
 2.00000E 02                2.72027E 01
 2.50000E 02                3.04489E 01
 3.00000E 02                3.34543E 01
 3.50000E 02                3.62149E 01
 4.00000E 02                3.87268E 01
 4.50000E 02                4.09857E 01
 5.00000E 02                4.29874E 01
 5.50000E 02                4.47276E 01
 6.00000E 02                4.62017E 01
 6.50000E 02                4.74050E 01
 7.00000E 02                4.83328E 01
 7.50000E 02                4.89801E 01
 8.00000E 02                4.93417E 01
 8.50000E 02                4.94125E 01
 9.00000E 02                4.91867E 01
 9.50000E 02                4.86591E 01
 1.00000E 03                4.78236E 01

READY
```

Fig. 4.31. Exemplu de folosire a limbajului BASIC la SD-8080

BIBLIOGRAFIE

1. DANCEA, I., *Microprocesoare*, Editura Dacia, Cluj-Napoca, 1979.
2. * * * *INTEL 8080 Microcomputer Systems User's Manual.*
3. * * * *INTEL 8080 Assembly Language Programming Manual.*
4. PEATMAN, J., *Microcomputer-based Design*, Mc Graw-Hill, London, 1977.
5. GREENE, B., *Application of Intel's 5 V EPROM and ROM family for Microprocessor Systems*, AP-30, Intel Corp.
6. GREENE, B., *Application of the Intel 2708 8K EPROM*, AP-17, Intel Corp.
7. OLIPHANT, J., *Designing Memory Systems with the Intel 2107A 4K RAM*, Intel Corp.
8. BARTHMAIER, J., *INTEL MULTIBUS Interfacing*, AP-28A, Intel Corp.
9. * * * *INTEL Data Catalog*, 1979, 1981.
10. * * * *Peripheral Design Handbook*, Intel Corp.
11. * * * *Memory Design Handbook*, Intel Corp.
12. ALTMAN, L., editor, *Microprocessors*, Electronic Book Series, culegere de articole din revista *Electronics* pe anii 1972—1975.
13. * * * *Mostek 1979 Memory Data Book and Designers' Guide.*
14. * * * *Motorola's 6800 vs Intel's 8080, a side-by-side Comparison*, Integrated Computer Systems, Belgia, 1977.
15. FARINA, M.V., *Programming in BASIC*, Prentice-Hall, Englewood Cliffs, 1968.
16. BLAKESLEE, T.R., *Digital Design with Standard MSI and LSI*, cap. 7—8, John Wiley and Sons, 1975.
17. BIEWER, M., *Microprocessor Architecture*, Pro-Log Microprocessor User's Guide, 1979/1980.
18. DODESCU, Gh., *Limbajul BASIC și aplicații*, EDP 1978, București.
19. SOUCEK, B., *Microprocessors and Microcomputers*, John Wiley and Sons Inc., 1976.
20. Mc CRACKEN, D.D., *A Guide to Intellec Microcomputer Development Systems*, Intel Corp., 1978.
21. * * * SDK-80 *System Development Kit*, Intel Corp., 1975.
22. * * * , *IMSAI 8080 Self-Contained System*, IMSAI Mfg. Comp., 1976.
23. * * *, *2650 Demo System*, Signetics SP 51 Application Note, Signetics Corp., 1976.
24. * * * *The ABC 1500 Adaptable Board Computer*, Signetics SP-55 Application Note, Signetics Corp., 1976.
25. * * * *„PIPBUG"* Signetics SP-50 Application Note, Signetics Corp., 1976.
26. ȚEPELEA, V.; KOVACS, A.; PURICE, E., *SD-8080, un sistem de dezvoltare pentru microprocesoare*, comunicare la sesiunea „I.T.C — 10 ani de activitate", București, 23—25 noiembrie 1978.
27. ȚEPELEA, V.; ROMAN, D., *Microcalculatoare-Structuri, Aplicații*, comunicare la sesiunea „Sisteme automate și informaționale în industrie", I.P.B., 28—30 aprilie 1975.
28. ȚEPELEA, V,; ROMAN, D., *Aspecte privind proiectare acu circuite integrate pe scară largă-Microprocesoare*, comunicare la sesiunea „Electrotehnica în cincinalul revoluției științifice și tehnice în România", I.C.P.E., București, 4—6 noiembrie 1976.

LIMBAJUL DE ASAMBLARE 8080. TEHNICI DE PROGRAMARE

5.1. GENERALITĂȚI

Un sistem de calcul poate efectua numai sarcinile care îi sînt transmise prin program. Programul este alcătuit dintr-o succesiune de instrucțiuni, prin care sînt specificate comenzile date calculatorului, în scopul obținerii unui rezultat util. Rezolvarea unei anumite probleme cu ajutorul calculatorului presupune exprimarea ei printr-un algoritm, adică indicarea pas cu pas a operațiilor care conduc la soluție. Aceste operații trebuie apoi codificate într-o succesiune de instrucțiuni pe care calculatorul le poate interpreta și executa.

Un calculator nu execută decît instrucțiunile exprimate intern sub forma unei sucesiuni de cifre binare (0 și 1). Programul în care instrucțiunile sînt scrise direct prin șiruri de cifre binare este denumit *program în limbaj-mașină* sau *în cod-calculator*.

Programarea în limbaj-mașină prezintă următoarele dificultăți:

1. programatorul trebuie să cunoască configurația binară a tuturor instrucțiunilor folosite;

2. întrucît programul de lucru este stocat în memoria calculatorului de unde el își extrage în secvență instrucțiunile, rezultă că programatorul trebuie să țină o evidență completă a ocupării memoriei, să cunoască în fiecare moment adresele libere, adresele ocupate, adresele unde a scris sau va scrie diferite instrucțiuni;

3. programul astfel elaborat este legat de mașina pentru care a fost scris.

Pentru a înlătura aceste inconveniente au fost create limbaje în care instrucțiunile sînt exprimate sub o formă simbolică, denumite *limbaje de programare*. Astfel, fiecărei instrucțiuni îi este atașat *un cod mnemonic* sau, pe scurt, *mnemonic*. Acest mod de reprezentare a instrucțiunilor-mașină se numește *limbaj simbolic* sau *limbaj de asamblare*.

Limbajul de asamblare permite utilizatorului să realizeze codificări simbolice ale reprezentării interne a memoriei unui calculator, ceea ce ușurează munca de programare. Reprezentarea simbolică a instrucțiunilor-mașină presupune anumite restricții, care determină regulile de scriere a programului în limbajul de asamblare. Aceste reguli formează *sintaxa programului*. Programul scris în limbaj simbolic se găsește într-o formă de reprezentare care nu permite să fie executat direct, el trebuie translatat în limbaj-mașină.

Acest proces se numește *traducere* sau *asamblare*. Forma inițială a programului, în limbaj de asamblare, se numește *program-sursă*, iar forma finală, în limbaj-mașină, se numește *program-obiect*.

Operația de traducere a programului-sursă în forma sa obiect este executată de un program special numit *asamblor*. În procesul de traducere controlul calculatorului este luat de către asamblor.

Program-sursă → ASAMBLOR → Program-obiect

Domeniul de definiție al asamblorului este mulțimea programelor-sursă, iar domeniul de valori, mulțimea programelor-obiect. Dacă un program-sursă supus operației de asamblare nu respectă regulile sintactice impuse de limbajul respectiv, iese din domeniul de definiție, nu se va mai obține un program-obiect, ci un mesaj de informare cu privire la erorile din programul-sursă. În mod obișnuit, fiecare instrucțiune din programul-sursă este translatată într-o singură instrucțiune din limbajul-mașină.

În afara codurilor mnemonice, cu un corespondent direct în limbajul-mașină, se mai folosesc și unele *cuvinte-cheie*, numite *pseudoinstrucțiuni*, care dirijează operația de asamblare și ajută pe programator să scrie cît mai ușor un program. Numărul acestora depinde de structura asamblorului.

Pseudoinstrucțiunile permit definirea în memorie a unor tabele de date, stabilirea adresei de început a unui program, rezervarea unor zone de memorie pentru diferite variabile folosite de program, stabilirea delimitatorilor unui program etc.

În general, asamblorul recunoaște drept program tot ce va fi cuprins între doi *delimitatori de program*, pe care îi notăm cu IP, începutul programului, și SP, sfîrșitul programului.

Pentru descrierea regulilor folosite în scrierea unui program în limbaj de asamblare, regulile sintactice ale unui program, este folosită notația BNF (Backus Normal Form) [6]. Astfel, noțiunea program este descrisă după cum urmează:

$$< \text{PROGRAM} > :: = \text{IP} < \text{ȘIR INSTRUCȚIUNI} > \text{SP}$$

Instrucțiunile din cadrul șirului vor fi despărțite prin *delimitatori de instrucțiune*, DI. Menționăm că delimitarea instrucțiunilor depinde de suportul extern de pe care se introduce programul-sursă în vederea asamblării.

Dacă programul-sursă este scris pe cartele perforate, va trebui ca pe o cartelă să se afle o singură instrucțiune. Dacă se folosește ca suport de intrare banda perforată, delimitatorul va fi grupul de coduri CR, Carriage Return, LF, Line Feed.

$$< \text{ȘIR INSTRUCȚIUNI} > :: =$$
$$= < \text{INSTR} > \mid < \text{INSTR} > \text{DI} < \text{ȘIR INSTRUCȚIUNI} >$$

O instrucțiune este compusă în principiu din patru cîmpuri: etichetă, cod-instrucțiune, operand, comentariu, la rîndul lor separate prin *delimitatori de cîmp*, DC:

$$< \text{INSTR} > :: =$$
$$= < \text{ETICHETĂ} > \text{DC} < \text{COD} > \text{DC} < \text{OPERAND} > \text{DC} < \text{COMENTARIU} >$$

Asamblorul trebuie să recunoască fiecare cîmp.

Eticheta este folosită, în general, pentru a putea referi adresa instrucțiunii pe care o precede. Într-o instrucțiune prezența etichetei nu este obligatorie, ea fiind folosită în funcție de necesitățile programatorului. De obicei, o etichetă începe cu o literă, iar lungimea sa variază de la asamblor la asamblor, în funcție de convențiile acestuia. De exemplu, un asamblor pentru microprocesorul 8080 admite simboluri cu maximum cinci caractere, iar asamblorul calculatorului FELIX C-256, cu maximum opt caractere. Eticheta poate fi definită după cum urmează:

< ETICHETĂ > :: = < SIMBOL >

< SIMBOL > :: =

= < LITERĂ > | < LITERĂ > < CARACTER > | < LITERĂ > < SIMBOL >

Codul mnemonic este simbolul asociat unei instrucțiuni-mașină sau este numele unei pseudoinstrucțiuni. De exemplu, pentru microprocesorul 8080:

< COD > :: = ACI | ADC | ADD | ADI | ... | XRI | XTHL | DB | DS | ... | ORG | SET

(§ 5.3.4 și § 5.3.5).

Operandul indică diverse adrese sau constante necesare cîmpului cod și poate fi definit după cum urmează:

<OPERAND> :: = <SIMBOL> | <CONSTANTĂ> |

<SIMBOL> <OP> <OPERAND>

<CONSTANTĂ> <OP> <OPERAND>

unde OP reprezintă un operator:

< OP > :: = + | − | * | /

Comentariul este alcătuit dintr-un șir de caractere alfanumerice și este utilizat de programator pentru descrierea operației executate de instrucțiunea în cauză. Acest cîmp este opțional în cadrul instrucțiunii și este definit după cum urmează:

< COMENTARIU > :: = < ȘIR DE CARACTERE >

< ȘIR DE CARACTERE > :: = < CARACTER > | < CARACTER >

< ȘIR DE CARACTERE >

< CARACTER > :: = A | B | C | D | ... | W | 0 | 1 | ... | 9

< CONSTANTĂ > :: = < CIFRĂ > | < CIFRĂ > < CONSTANTĂ >

< CIFRĂ > :: = 0 | 1 | ... | 9

< LITERĂ > :: = A | B | C | ... | W

Pentru a executa operația de translatare asamblorul execută două treceri peste programul-sursă.

În prima trecere este creat *tabelul de simboluri*, rezultat din simbolurile ce apar în cîmpul etichetă, este asignată memoria necesară programului și se face o analiză sintactică a programului, prin care se stabilește în ce măsură au fost respectate regulile limbajului de asamblare.

În trecerea a doua se obține programul-obiect.

Există posibilitatea ca în timpul primei treceri să se facă o copie a programului-sursă pe o memorie externă, a doua trecere executîndu-se fără intervenția operatorului.

Pentru a putea fi lansat în execuție, programul-obiect, rezultat în urma operației de asamblare pe un suport extern (bandă perforată, memorie externă etc.), este încărcat în memoria calculatorului cu ajutorul unui *program de încărcare.*

5.2. ARHITECTURA MICROPROCESORULUI 8080

Pentru a putea scrie un program în limbajul de asamblare 8080 programatorul trebuie să cunoască structura generală a microprocesorului 8080 și a microcalculatorului pe care se lucrează (realizat pe baza acestuia). În capitolul 4 este descrisă realizarea unui astfel de microcalculator din punct de vedere hardware, iar în figura 5.1 este prezentată organizarea internă a microprocesorului 8080.

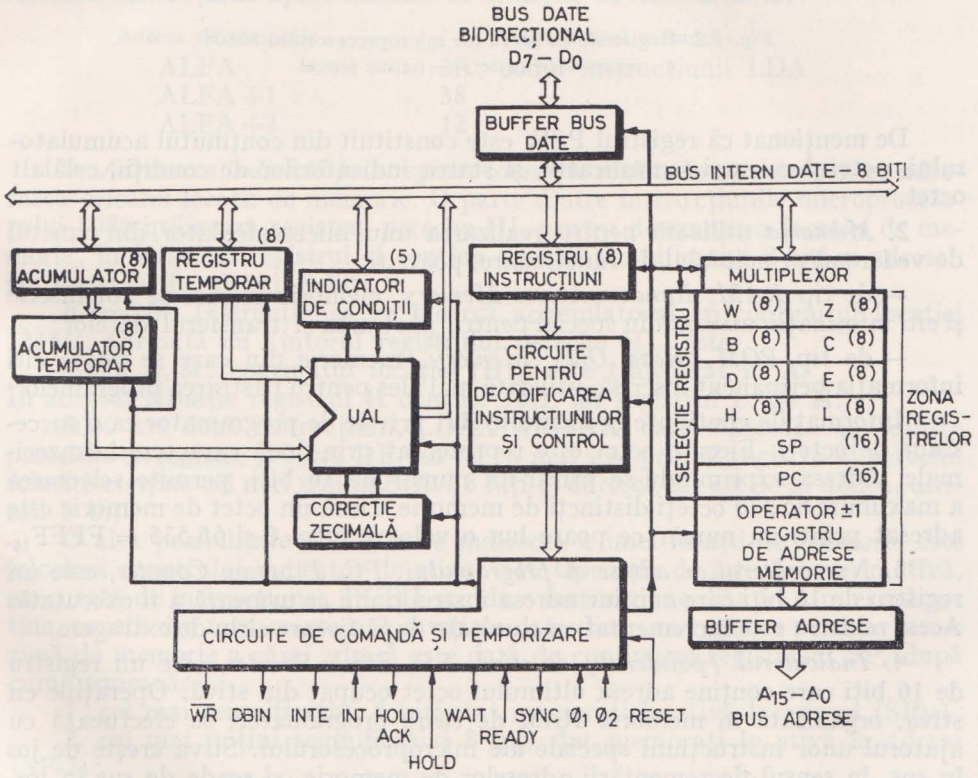


Fig. 5.1. Organizarea microprocesorului 8080

Pentru programator un asemenea microcalculator se compune din următoarele părți:

1. *Registrele de lucru* cu care se efectuează toate operațiile asupra datelor și care constituie un mijloc de adresare a memoriei. 8080 pune la dispoziția programatorului un registru acumulator de 8 biți și 6 registre de lucru de cîte 8 biți. Aceste registre sînt referite cu ajutorul numerelor întregi 0, 1, 2, 3, 4, 5 și 7 (fig. 5.2 *a*); prin convenție ele pot fi apelate cu ajutorul literelor *B*, *C*, *D*, *E*, *H*, *L* și, respectiv, *A*. Registrele pot fi folosite și ca registre pereche, cu lungimea de 16 biți, fiind referite cu ajutorul literelor *B* (BC), *D* (DE), *H* (HL) și *PSW* (fig. 5.2 *b*).

Registrul B →	0	1	← Registrul C	
Registrul D →	2	3	← Registrul E	
Registrul H →	4	5	← Registrul L	
	7		← Registrul A (Acumulatorul)	

a

Registrul pereche B →	0	1
Registrul pereche D →	2	3
Registrul pereche H →	4	5
Registrul pereche PSW →	7	F

b

Fig. 5.2. Registrele de lucru ale microprocesorului 8080:
a — registre individuale; *b* — registre pereche

De menționat că registrul PSW este constituit din conținutul acumulatorului, octetul cel mai semnificativ, și starea indicatorilor de condiții, celălalt octet.

2. *Memoria* utilizată pentru realizarea unui microcalculator, din punctul de vedere al principiului de funcționare, poate fi:

— de tip *RAM*, *Random Access Memory*, memorie în care se pot înscrie și citi informații, utilizată în special pentru păstrarea și transferul datelor;

— de tip *ROM*, *Read Only Memory*, memorie din care se poate citi informația permanent înscrisă, utilizată mai ales pentru păstrarea programelor.

Informațiile conținute în memorie sînt privite de programator ca o succesiune de octeți. Fiecare octet este reprezentat prin două caractere hexazecimale. Adresa exprimîndu-se printr-un număr de 16 biți, permite selectarea a maximum 65 536 octeți distincți de memorie. Deci, un octet de memorie este adresat printr-un număr ce poate lua o valoare între 0 și 65 535 = $FFFF_{16}$.

3. *Numărătorul de adrese al programului*, *PC*, *Program Counter*, este un registru de 16 biți care conține adresa instrucțiunii ce urmează a fi executată. Acest registru este incrementat cu o unitate după fiecare ciclu de extragere.

4. *Indicatorul (pointerul) de stivă*, *SP*, *Stack Pointer*, este un registru de 16 biți care conține adresa ultimului octet ocupat din stivă. Operațiile cu stiva, organizată în memoria RAM de către programator, se efectuează cu ajutorul unor instrucțiuni speciale ale microprocesorului. Stiva crește de jos în sus, în sensul decrementării adreselor de memorie, și scade de sus în jos, în sensul incrementării adreselor de memorie.

5. Partea de *intrare/ieşire* reprezintă legătura microcalculatorului cu exteriorul. Microprocesorul 8080 permite conectarea a pînă la 256 dispozitive de intrare şi 256 dispozitive de ieşire. Fiecare dispozitiv de I/E comunică cu calculatorul prin intermediul acumulatorului şi are ca adresă un număr cuprins între 0 şi 255 ce nu poate fi modificat de programator.

5.2.1. MODURI DE ADRESARE

Prin mod de adresare se înţelege felul în care se determină adresa unei locaţii ce urmează a participa într-o operaţie. Microprocesorul 8080 permite următoarele moduri de adresare:

— *Adresarea directă* se referă la faptul că octeţii 2 şi 3 ai instrucţiunii conţin adresa locaţiei de memorie în care este amplasat operandul. Biţii cei mai semnificativi ai adresei sînt conţinuţi în cel de-al treilea octet, iar biţii cei mai puţin semnificativi în cel de-al doilea octet.

Exemplul 5—1. Pentru a încărca în acumulator conţinutul locaţiei de memorie 1238H se execută următoarea instrucţiune:

ALFA: LDA 1238H

Această instrucţiune apare înscrisă în memorie în felul următor:

Adresa de memorie	Conţinutul memoriei
ALFA	3A; codul instrucţiunii LDA
ALFA+1	38
ALFA+2	12

— *Adresarea indirectă prin registre* foloseşte registrele pereche pentru adresarea oricărei locaţii de memorie. O parte dintre instrucţiunile microprocesorului 8080 utilizează registrul pereche HL pentru desemnarea locaţiei de memorie; în acest caz registrul H conţine cei mai semnificativi 8 biţi ai adresei, iar registrul L, ultimii 8 biţi.

Exemplu. Instrucţiunea ce încarcă acumulatorul cu conţinutul locaţiei 1238H, adresată cu ajutorul registrului pereche HL, este:

MØV A, M; conţinutul locaţiei (HL) este transferat în A.

În această situaţie registrul H conţine 12H, iar registrul L, 38 H.

8080 are două instrucţiuni, LDAX şi STAX, care utilizează una dintre perechile de registre BC sau DE. La fel ca mai sus, primul registru al perechii folosite conţine cei mai semnificativi 8 biţi ai adresei, iar cel de-al doilea, ultimii 8 biţi.

O altă posibilitate de adresare indirectă a unei locaţii de memorie este folosirea registrului indicator de stivă, SP. Operaţia de introducere în stivă, executată de instrucţiunea PUSH sau de o instrucţiune de apel a unei subrutine, constă din memorarea a 2 octeţi de date dintr-un registru pereche într-o zonă de memorie a cărei adresă este dată de conţinutul registrului SP, după cum urmează:

1. cei mai semnificativi 8 biţi sînt memoraţi în stivă la adresa (SP)-1;
2. cei mai puţini semnificativi 8 biţi sînt memoraţi în stivă la adresa (SP-2);
3. conţinutul registrului SP este decrementat cu 2, (SP) ← (SP) — 2.

Exemplul 5-2. Presupunem că (SP) = 3FD0H, (D) = 27H și (E)=7EH. După executarea operației de memorare în stivă a dublului registru DE, conținutul acestora va fi: (SP)=3FCEH, (D)=27H și (E)=7EH. Conținutul stivei este:

Anterior operației de salvare	Adresă memorie	După operația de salvare
FF	3FCD	FF
FF	3FCE	7E
FF	3FCF	27
FF	3FD0	FF

Operația de extragere din stivă, executată de instrucțiunea PØP sau de o instrucțiune de revenire dintr-o subrutină, constă din transferarea a 2 octeți de la adresa dată de conținutul registrului SP într-un registru pereche, după cum urmează:

1. în cel de-al doilea registru al perechii, sau în cei mai puțin semnificativi 8 biți ai registrului PC, se încarcă conținutul locației de memorie indicată de conținutul registrului SP;
2. în primul registru, sau în cei mai semnificativi 8 biți ai registrului PC, se încarcă conținutul locației de memorie cu adresa (SP)+1;
3. conținutul registrului SP este incrementat cu 2, (SP) ← (SP) + 2.

Exemplul 5-3. Se presupune că (H) = 0FFH, (L) = 0FFH, SP=3FC2H și conținutul vîrfului de stivă este 85H la adresa 3FC2H, respectiv 07H la 3FC3H. După execuția operației de transfer a conținutului vîrfului de stivă în registrul pereche HL vom avea: (SP)=3FC4H, (H)=07H, (L)= 85H. Conținutul stivei este:

Anterior operației de extragere din stivă	Adresă memorie	După operația de extragere din stivă
FF	3FC1	FF
85	3FC2	85
07	3FC3	07
FF	3FC4	FF

— *Adresarea imediată* semnifică faptul că instrucțiunea conține operandul, un octet sau doi octeți în funcție de lungimea acestuia, în octetul sau octeții ce urmează codului de operație.

— *Adresarea registrelor* se referă la instrucțiunile al căror cod de operație specifică unul sau două registre generale ce intervin în operația de prelucrare indicată de instrucțiune. Exemplu: instrucțiunile de transfer al informației între registre: MØV r_1, r_2.

— *Adresarea implicită* se referă la instrucțiunile al căror cod de operație implică unul sau mai multe registre ce conțin operanzi. Exemplu: utilizarea acumulatorului în operațiile aritmetice și logice.

— *Adresarea combinată.* O parte dintre instrucțiuni au mai mult de un operand, ca de exemplu instrucțiunile aritmetice. În acest caz două tipuri de adresare pot fi combinate.

5.2.2. INDICATORII DE CONDIȚII

Microprocesorul 8080 are 5 indicatori de condiții care furnizează informații despre rezultatul operațiilor aritmetice și logice ce se execută asupra datelor. Acești indicatori sînt:

1. *Indicatorul de transport — CY, Carry*, pune în evidență un transport din bitul cel mai semnificativ al rezultatului, iar starea sa poate fi testată prin program. CY este afectat de operațiile de adunare, scădere, comparație, rotire, precum și de operațiile logice.

Exemplu. După execuția următoarei operații de adunare:

$$\begin{array}{r} AE = 10101110 \\ +\ 74 = 01110100 \\ \hline 122\ \ 1]\ 00100010 \end{array}$$

este generat un transport în afara bitului cel mai semnificativ al rezultatului, ceea ce va determina poziționarea indicatorului CY, CY = 1.

2. *Indicatorul de transport auxiliar — AC, Auxiliary Carry*, indică un transport din bitul 3 în bitul 4. Starea indicatorului AC este folosită intern la execuția instrucțiunii DAA și nu poate fi testată prin program.

Exemplu. După execuția operației de adunare:

$$\begin{array}{r} 29 = 00101001 \\ +78 = 01111000 \\ \hline A1 = 10100001 \end{array}$$

se produce un transport din bitul 3 în bitul 4 al rezultatului, ceea ce va determina poziționarea indicatorului AC, AC = 1.

3. *Indicatorul de semn — S, Sign.* În situația în care un octet este interpretat ca un număr cu semn în complement față de 2, bitul cel mai semnificativ, al 8-lea, va fi considerat ca bit de semn. Prin convenție, bitul de semn are valoarea 0 pentru numerele pozitive și 1 pentru numerele negative. S este poziționat dacă rezultatul unei operații aritmetice sau logice are bitul cel mai semnificativ egal cu 1, *Minus*. În caz contrar S va fi șters, S = 0, *Positive*.

4. *Indicatorul de paritate — P, Parity.* P este poziționat, P = 1, *Parity Even*, dacă suma modulo 2 a biților rezultatului unei operații aritmetice sau logice este 0, adică rezultatul are un număr par de unități; în caz contrar P va fi șters, P = 0, *Parity Odd*.

5. *Indicatorul de zero — Z, Zero.* Z este poziționat, Z = 1, dacă rezultatul generat în acumulator după execuția unei instrucțiuni aritmetice sau logice este 0; în caz contrar va fi șters, Z = 0.

Indicatorii Z, S, P și CY sînt testabili prin program (vezi instrucțiunile *Jcc*, *Rcc* și *Ccc*).

5.3. SETUL DE INSTRUCȚIUNI 8080

5.3.1. FORMATUL INSTRUCȚIUNILOR

Datorită structurii interne a microprocesorului 8080, memoria unui microcalculator realizat pe baza acestuia este organizată pe cuvinte cu lungimea de 8 biți; oricărui cuvînt îi corespunde o adresă de 16 biți. În fiecare octet bitul din extremitatea dreaptă, D_0, este considerat cel mai puțin semnificativ, iar bitul din extremitatea stîngă, D_7, cel mai semnificativ (fig. 5.3*a*).

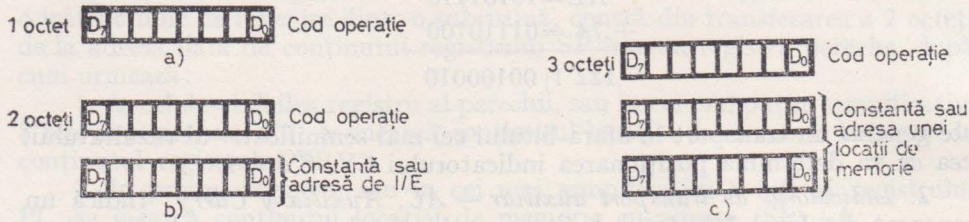


Fig. 5.3. Formatul instrucțiunilor microprocesorului 8080

În formă internă, fiecare instrucțiune este reprezentată printr-o succesiune de biți. Instrucțiunile microprocesorului 8080 pot avea lungimea de un octet, doi octeți sau trei octeți. Într-o instrucțiune multioctet, ei sînt memorați în locații succesive de memorie. Adresa primului octet este întotdeauna și adresa instrucțiunii, iar el reprezintă codul operației. Pentru instrucțiunile pe doi octeți ultimul dintre aceștia poate fi o adresă de I/E sau o constantă, iar în cazul instrucțiunilor pe trei octeți ultimii doi pot reprezenta o adresă de memorie sau o constantă (fig. 5.3).

5.3.2. SINTAXA INSTRUCȚIUNILOR

Instrucțiunile limbajului de asamblare 8080 respectă regulile de formare general admise pentru limbajele simbolice (vezi § 5.1). O instrucțiune are patru părți distincte, denumite cîmpuri, după cum urmează:

1. *eticheta* utilizată pentru a referi adresa instrucțiunii pe care o reprezintă ;
2. *codul* indică operația ce se execută;
3. *operandul* indică diverse adrese sau date necesare cîmpului-cod;
4. *comentariul*, utilizat de programator pentru descrierea operației executate de instrucțiunea în cauză.

5.3.2.1. Cîmpul-etichetă

Cîmpul-etichetă este denumit şi cîmp de adresă. În limbaj de asamblare fiecare instrucţiune poate primi o etichetă, însă nu este necesar ca toate instrucţiunile unui program să fie etichetate.

Eticheta are rolul de a identifica în cadrul programului simbolic instrucţiunile care reprezintă destinaţia unor operaţii de salt; asamblorul le transformă în adrese în cadrul programului-obiect. Unul din avantajele utilizării etichetelor îl constituie posibilitatea modificării programului, prin introducerea sau eliminarea unor instrucţiuni, fără a fi necesară rescrierea sa; de asemenea, folosirea lor uşurează înţelegerea programului. Lungimea unei etichete este limitată la cinci caractere; primul caracter al etichetei trebuie să fie o literă a alfabetului sau unul dintre caracterele speciale „?" sau „@". După ultimul caracter al etichetei trebuie să urmeze caracterul „:", acesta fiind considerat ca un separator între cîmpul-etichetă şi cîmpul cod-instrucţiune. Simbolurile ce reprezintă coduri instrucţiune, numele registrelor şi numele pseudoinstrucţiunilor nu pot fi utilizate ca etichete.

Dacă programul-sursă este scris pe cartele, eticheta va începe din prima coloană. Atunci cînd programul-sursă este scris pe bandă perforată, eticheta va fi perforată după separatorul de instrucţiune, grupul CR, LF. În situaţia în care eticheta lipseşte dintr-o instrucţiune aceasta este înlocuită cu cel puţin un blanc.

5.3.2.2. Cîmpul-cod

Acest cîmp conţine un simbol denumit mnemonicul instrucţiunii, care identifică operaţia. El poate fi corespondentul codului unei instrucţiuni-maşină sau mnemonicul unei pseudoinstrucţiuni specifice limbajului de asamblare. Între acesta şi cîmpul-etichetă poate să existe un număr oarecare de blancuri. Codul este separat de următorul cîmp al instrucţiunii prin cel puţin un blanc.

5.3.2.3. Cîmpul-operand

Informaţiile furnizate de acest cîmp împreună cu codul definesc operaţia ce trebuie executată de instrucţiunea în cauză. În funcţie de cîmpul cod, operandul poate fi absent sau poate conţine unul sau două argumente, separate prin virgulă.

Cîmpul-operand poate specifica: un registru simplu, un registru pereche, o dată imediată, o adresă de memorie (2 octeţi) sau o adresă de I/E (un octet).

În limbajul de asamblare 8080 cîmpul-operand poate fi exprimat în 9 moduri, după cum urmează:

1. O *constantă hexazecimală*, ce trebuie să înceapă cu o cifră 0÷9 şi să se termine cu litera H.
2. O *constantă zecimală*, ce poate fi urmată opţional de litera D.
3. O *constantă octală*, ce trebuie să se termine cu una din literele O sau Q.

4. O *constantă binară*, ce trebuie să se termine cu litera B.

5. *Numărătorul de adresă-program*, specificat cu ajutorul caracterului $ ce reprezintă adresa instrucțiunii curente.

Exemplu:

```
GAMA: JMP   $+6     ;INSTRUCȚIUNE PE 3 OCTEȚI
      CALL  AAI     ;INSTRUCȚIUNE PE 3 OCTEȚI
      MØV   A,B     ;INSTRUCȚIUNEA LA CARE SE EXECUTĂ
                    ;SALTUL.
```

Prima instrucțiune va poziționa, după executarea sa, numărătorul de adresă-program, PC, la o adresă mai mare cu 6 octeți decît adresa curentă. Executarea programului va continua cu instrucțiunea MØV A,B.

6. O *constantă ASCII*, întîlnită și sub denumirea de *literal*, constă din unul sau mai multe caractere ASCII incluse între ghilimele simple.

Exemplu:

```
CARAC: MVI  D,'*'   ;ÎNCARCĂ REG. D CU CODUL ASCII AL
                    ;CARACTERULUI ASTERISC — 2AH.
```

7. O *etichetă* căreia asamblorul i-a atribuit o valoare numerică.

Corespondența *registru-valoare numerică* recunoscută de limbajul de asamblare 8080 este dată în figura 5.4.

B—0	D—2	H—4	M—6	SP—6
C—1	E—3	L—5	PSW—6	A—7

Fig. 5.4. Corespondența dintre simbolurile atribuite registrelor de lucru și valoarea numerică corespunzătoare.

8. O *etichetă* ce apare în cîmpul respectiv al instrucțiunilor unui program.

9. *Expresii aritmetice și logice.* Acestea utilizează toate tipurile de date descrise mai sus, constituind *operanzii expresiei.* Operanzi sînt legați cu ajutorul operatorilor aritmetici: + (adunare), — (scădere), * (înmulțire), /(împărțire), MOD (modulo), sau logici: NOT, AND, OR, XOR, de deplasare SHR, SHL și cu ajutorul parantezelor stînga și dreapta. Lungimea datelor (operanzilor) luate în considerare de asamblor este de 16 biți.

Operatorii aritmetici realizează, în ordinea enumerării lor, adunarea, scăderea, înmulțirea, împărțirea întreagă și, respectiv, calculul restului împărțirii efectuate între operanzii lor. Operatorii logici acționează bit cu bit și produc complementarea, produsul logic, suma logică și, respectiv, suma modulo 2 a argumentelor.

Operatorii SHL și SHR produc deplasarea liniară a primului operand spre stînga, respectiv dreapta, cu un număr de poziții egal cu valoarea celui de-al doilea operand. În partea opusă deplasării se introduce un număr de zerouri egal cu numărul de deplasări.

Ordinea în care sînt executate operațiile aritmetice și logice dintr-o expresie este:

1. expresiile dintre paranteze;
2. *, /, MØD, SHL, SHR;
3. +, —;

4. NØT;
5. AND;
6. ØR, XØR.

Operatorii MØD, SHL, SHR, NØT, AND, ØR și XØR trebuie să fie separați de operanzi cu cel puțin un blanc.

5.3.2.4. Cîmpul-comentariu

Cîmpul-comentariu este facultativ. El permite programatorului să introducă scurte explicații cu referire la o instrucțiune sau la un grup de instrucțiuni. În acest fel programul devine mai inteligibil. Pentru ca asamblorul să recunoască un cîmp comentariu, acesta din urmă va trebui să fie precedat de caracterul „;". Între cîmpul precedent și „;", începutul unui comentariu, poate să apară un număr variabil de blancuri.

Limbajul de asamblare 8080 permite inserarea unor linii de comentariu în cadrul programului, cu condiția ca acestea să înceapă cu caracterul „;".

5.3.3. REPREZENTAREA INTERNĂ A DATELOR

În orice sistem de calcul, deci și într-un microcalculator bazat pe 8080, există două categorii de informații, instrucțiunile și datele, reprezentate intern prin succesiuni de cifre binare. Codificarea instrucțiunilor este impusă de microprocesor și nu poate fi modificată de utilizator. Atunci cînd un program este scris în limbaj de asamblare utilizatorul nu trebuie să-și facă nici un fel de probleme privind codificarea instrucțiunilor, deoarece aceasta se va face întotdeauna corect în urma operației de asamblare a programului-sursă. În consecință, utilizatorul va trebui să fie atent numai la modul de reprezentare al datelor necesare în execuția programului. Datele pot fi numerice și alfanumerice.

Codificarea datelor alfanumerice este necesară pentru a putea memora mesaje sau caractere speciale și se face în format de 8 biți. Al 8-lea bit din cod, cel mai semnificativ, poate fi utilizat pentru verificarea parității. Pentru 8080 s-a adoptat folosirea codului ASCII (American Standard Code for Information Interchange). În funcție de instrucțiune, în cîmpul operand pot apărea date numerice reprezentate pe unul sau doi octeți. În cazul instrucțiunilor aritmetice se presupune că datele sînt reprezentate în *complement față de 2*.

Cînd un număr de un octet este considerat ca *număr cu semn în complement față de 2*, primii 7 biți reprezintă mărimea numărului, iar bitul cel mai semnificativ, al 8-lea, este interpretat ca semn, 0 pentru numere pozitive și 1 pentru numere negative.

Rezultă că gama numerelor pozitive ce pot fi reprezentate printr-un număr cu semn în complement față de 2 este cuprinsă între 0 și 127:

$$0 = 0\,0\,0\,0\,0\,0\,0\,0 = 0H$$
$$1 = 0\,0\,0\,0\,0\,0\,0\,1 = 1H$$
$$\vdots$$
$$127 = 0\,1\,1\,1\,1\,1\,1\,1 = 7FH$$

La schimbarea semnului unui număr reprezentat în complement față de 2 se aplică următoarele reguli:

1. se formează complementul față de 1 al numărului, prin schimbarea fiecărei cifre binare cu complementul său (0 cu 1 și 1 cu 0);
2. se adună o unitate în poziția cea mai puțin semnificativă a rezultatului de la punctul precedent și se ignoră transportul din bitul cel mai semnificativ.

Exemplu. Reprezentarea în complement față de 2 a numărului — 1CH:

$$+1CH = 0\,0\,0\,1\,1\,1\,0\,0$$
$$1. \quad 1\,1\,1\,0\,0\,0\,1\,1$$
$$2. \quad 1\,1\,1\,0\,0\,1\,0\,0$$

Deci, reprezentarea în complement față de 2 a numărului — 1CH este 0E4H. Bitul 8 egal cu 1 indică faptul că este vorba de un număr negativ.

Gama numerelor negative ce pot fi reprezentate printr-un număr cu semn în complement față de 2 este cuprinsă între —1 și — 128:

$$-1 = 1\,1\,1\,1\,1\,1\,1 = 0FFH$$
$$-2 = 1\,1\,1\,1\,1\,1\,0 = 0FEH$$
$$\vdots$$
$$-127 = 1\,0\,0\,0\,0\,0\,1 = 81H$$
$$-128 = 1\,0\,0\,0\,0\,0\,0 = 80H$$

Dacă un octet este interpretat ca un *număr fără semn* el se consideră pozitiv și va fi cuprins între 0 și 255:

$$0 = 0\,0\,0\,0\,0\,0\,0\,0 = 0H$$
$$1 = 0\,0\,0\,0\,0\,0\,0\,1 = 1H$$
$$\vdots$$
$$255 = 1\,1\,1\,1\,1\,1\,1\,1 = 0FFH$$

5.3.4. DESCRIEREA INSTRUCȚIUNILOR 8080

Microprocesorul 8080 are un set de 78 de instrucțiuni de lungime variabilă, 1, 2, sau 3 octeți. Primul octet al fiecărei instrucțiuni conține *codul-operației*. Din cele 256 combinații disponibile pentru codificarea instrucțiunilor, 8080 folosește numai 244 (tabelul 5.1). Instrucțiunile microprocesorului 8080 sînt prezentate sintetic în continuare.

NOTAȚII

( ) — conținutul unui registru;

(( )) — conținutul unei locații de memorie, a cărei adresă este dată de conținutul unui registru pereche, *rp*, PC sau SP.

Mnemonic	Descrierea instrucțiunii		Indicatori afectați	Nr. stări	Obs.
1	2		3	4	5
MOV r_1, r_2	Move register to register	$(r_1) \leftarrow (r_2)$		5	1,2
MOV M, r	Move register to memory	$((H)(L)) \leftarrow (r)$		7	3
MOV r, M	Move memory to register	$(r) \leftarrow ((H)(L))$		7	3
ADD r	Add register to A	$(A) \leftarrow (A) + (r)$	Z, S, P, CY, AC	4	3,4
ADC r	Add register to A with carry	$(A) \leftarrow (A) + (r) + (CY)$	Z, S, P, CY, AC	4	3
SUB r	Subtract register from A	$(A) \leftarrow (A) - (r)$	Z, S, P, CY, AC	4	3,5
SBB r	Subtract register from A with borrow	$(A) \leftarrow (A) - (r) - (CY)$	Z, S, P, CY, AC	4	3
ANA r	And register with A	$(A) \leftarrow (A) \wedge (r)$	Z, S, P, CY=0	4	3,6
XRA r	Exclusive Or register with A	$(A) \leftarrow (A) \oplus (r)$	Z, S, P, CY=0	4	3,7
ORA r	Or register with A	$(A) \leftarrow (A) \vee (r)$	Z, S, P, CY=0	4	3,8
CMP r	Compare register with A	$(A) - (r)$	Z,S,P, CY, AC	4	3,9
ADD M	Add memory to A	$(A) \leftarrow (A) + ((H)(L))$	Z, S, P, CY, AC	7	
ADC M	Add memory to A with carry	$(A) \leftarrow (A) + ((H)(L)) + (CY)$	Z, S, P, CY, AC	7	
SUB M	Subtract memory from A	$(A) \leftarrow (A) - ((H)(L))$	Z, S, P, CY, AC	7	
SBB M	Subtract memory from A with borrow	$(A) \leftarrow (A) - ((H)(L)) - (CY)$	Z, S, P, CY. AC	7	
ANA M	And memory with A	$(A) \leftarrow (A) \wedge ((H)(L))$	Z, S, P, CY=0	7	
XRA M	Exclusive Or memory with A	$(A) \leftarrow (A) \oplus ((H)(L))$	Z, S, P, CY=0	7	
ORA M	Or memory with A	$(A) \leftarrow (A) \vee ((H)(L))$	Z, S, P, CY=0	7	
CMP M	Compare memory with A	$(A) - ((H)(L))$	Z, S, P, CY, AC	7	
INR r	Increment register	$(r) \leftarrow (r) + 1$	Z,S,P,AC	5	
DCR r	Decrement register	$(r) \leftarrow (r) - 1$	Z, S, P, AC	5	
INR M	Increment memory	$((H)(L)) \leftarrow ((H)(L)) + 1$	Z, S, P, AC	10	
DCR M	Decrement memory	$((H)(L)) \leftarrow ((H)(L)) - 1$	Z, S, P, AC	10	
MVI r, *D8*	Move to register immediate	$(r) \leftarrow D8$		7	
MVI M, *D8*	Move to memory immediate	$((H)(L)) \leftarrow D8$		10	
ADI *D8*	Add immediate to A	$(A) \leftarrow (A) + D8$	Z, S, P, CY, AC	7	10
ACI *D8*	Add immediate to A with carry	$(A) \leftarrow (A) + D8 + (CY)$	Z, S, P, CY, AC	7	
SUI *D8*	Subtract immediate from A	$(A) \leftarrow (A) - D8$	Z, S, P, CY, AC	7	
SBI *D8*	Subtract immediate from A with borrow	$(A) \leftarrow (A) - D8 - (CY)$	Z, S, P, CY, AC	7	

1	2		3	4	5
ANI *D8*	And immediate with A	(A) ← (A) ∧ *D8*	Z, S, P, CY=0	7	
XRI *D8*	Exclusive Or immediate with A	(A) ← (A) ⊕ *D8*	Z, S, P, CY=0	7	
ORI *D8*	Or immediate with A	(A) ← (A) ∨ *D8*	Z, S, P, CY=0	7	
CPI *D8*	Compare immediate with A	(A) − *D8*	Z, S, P, CY, AC	7	
RLC	Rotate A left	$A_{n+1} \leftarrow A_n$, $A_0 \leftarrow A_7$, (CY) ← A_7	CY	4	
RRC	Rotate A right	$A_n \leftarrow A_{n+1}$, $A_7 \leftarrow A_0$ (CY) ← A_0	CY	4	
RAL	Rotate A left through carry	$A_{n+1} \leftarrow A_n$, (CY)←A_7 A_0 ← (CY)	CY	4	
RAR	Rotate A right through carry	$A_n \leftarrow A_{n+1}$, (CY)←A_0 A_7 ← (CY)	CY	4	
LXI *rp, D16*	Load immediate register pair *rp*	(*rp*) ← *D16* (*rp*=B, D, H sau SP)		10	11, 12
INX *rp*	Increment register pair *rp*	(*rp*) ← (*rp*) + 1 (*rp*=B,D,H sau SP)		5	
DCX *rp*	Decrement register pair *rp*	(*rp*) ← (*rp*) − 1 (*rp*=B,D, H sau SP)		5	
PUSH *rp*	Push register pair *rp* on stack	((SP) − 1)←(*rp*$_H$), ((SP) −2 ← (*rp*$_L$), (SP) ← (SP) − 2 (*rp*=B,D sau H)		11	17
PUSH PSW	Push A and Flags on stack	((SP) − 1) ← (A), ((SP) − 2) ← (F), (SP) ← (SP) − 2		11	13
POP *rp*	Pop register pair off stack	(*rp*$_L$) ← ((SP)), (*rp*$_H$) ← ((SP)+ 1), (SP) ← (SP) + 2 (*rp* = B, D sau H)		10	17
POP PSW	Pop A and Flags off stack	(F) ← ((SP)), (A) ← ((SP)+ 1), (SP) ← (SP) + 2	Z, S, P, CY, AC	10	
DAD *rp*	Add *rp* to HL	(HL) ← (HL)+(*rp*) (*rp*=B, D,H sau SP)	CY	10	14
LDA *Adr*	Load A direct	(A) ← (*Adr*)		13	11
STA *Adr*	Store A direct	(*Adr*) ← (A)		13	
LDAX *rp*	Load A indirect	(A) ← ((*rp*))		7	
STAX *rp*	Store A indirect	((*rp*)) ← (A) (*rp* = B sau D)		7	
LHLD *Adr*	Load HL direct	(L) ← (*Adr*), (H) ← (*Adr* + 1)		16 16	
SHLD *Adr*	Store HL direct	(*Adr*) ← (L), (*Adr* + 1) ← (H)		16	
XCHG	Exchange DE, HL registers	(H) ↔ (D), (L) ↔ (E)		4	
XTHL	Exchange top of stack, HL	(L) ↔ ((SP)) (H) ↔ ((SP) + 1)		18	
SPHL	HL to stack pointer	(SP) ← (HL)		5	15
CMC	Complement carry	(CY) ← $(\overline{CY})$	CY	4	
STC	Set carry	(CY) ← 1	CY	4	
CMA	Complement A	(A) ← $(\bar{A})$		4	

1	2		3	4	5
DAA	Decimal adjust A	Dacă $(A_0 \div A_3) > 9$ sau $(AC) = 1$ atunci $(A) \leftarrow (A) + 6$; dacă $(A_4 \div A_7) > 9$ sau $(CY) = 1$ atunci $(A) \leftarrow (A) + 6 * 2^4$	Z, S, P, AC, CY	4	16
NOP	No-operation			4	
HLT	Halt			7	18
EI	Enable Interrupts	(INTE) ← 1		4	
DI	Disable Interrupts	(INTE) ← 0		4	
IN *Exp*	Input	(A) ← (*Exp*)		10	19
OUT *Exp*	Output	(*Exp*) ← (A)		10	20
PCHL	HL to program counter	(PC) ← (HL)		5	
JMP *Adr*	Jump unconditional	(PC) ← *Adr*		10	
J*cc Adr*	Jump on condition *cc*	Dacă *cc* adevărată (PC) ← *Adr* altfel (PC) ← (PC) + 3		10	21
CALL *Adr*	Call unconditional	$((SP) - 1) \leftarrow (PC_H)$, $((SP) - 2) \leftarrow (PC_L)$, $(SP) \leftarrow (SP) - 2$, $(PC) \leftarrow Adr$		17	17
C*cc Adr*	Call on condition *cc*	Dacă *cc* adevărată execută CALL *Adr* altfel (PC) ← (PC) + + 3		17/11	21
RET	Return	$(PC_L) \leftarrow ((SP))$, $(PC_H) \leftarrow ((SP) + 1)$, $(SP) \leftarrow (SP) + 2$		10	17
R*cc*	Return on condition *cc*	Dacă *cc* adevărată se execută RET altfel (PC) ← (PC) + + 1		11/5	21
RST *Exp*	Restart	$((SP) - 1) \leftarrow (PC_H)$ $((SP) - 2) \leftarrow (PC_L)$, $(SP) \leftarrow (SP) - 2$, $(PC) \leftarrow Exp \times 8$		11	17

Observații:

1. r_1, r_2 = A, B, C, D, E, H sau L.

2. Orice instrucțiune de tipul MØV X, X este considerată ca *operație nulă* și poate fi înlocuită cu instrucțiunea NØP.

3. r = A, B, C, D, E, H sau L.

4. Dacă (A) = 6CH și (B) = 2EH, atunci după execuția instrucțiunii ADD B

$$\begin{array}{r} 6CH = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0 \\ +2EH = 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\ \hline 9AH = 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \end{array}$$

(A) = 9AH, (B) = 2EH, Z = 0, CY = 0, AC = 1, S = 1 și P = 1.

5. Operația de scădere se execută adunînd conținutul acumulatorului cu complementul față de 2 al scăzătorului. Indicatorul de transport, CY, după execuția operației de scădere sau comparare va fi egal cu transportul negat din bitul cel mai semnificativ al rezultatului.

Valoarea indicatorului CY trebuie interpretată în funcție de modul de reprezentare internă a celor două mărimi cu care se operează:

a. numere fără semn $0 \leqslant N \leqslant 255$;

b. numere cu semn $-128 \leqslant N \leqslant 127$.

Aceasta înseamnă că:

a. CY = 0 dacă (A) $\geqslant$ (*r*) și CY = 1 dacă (A) < (*r*);

b. CY = 0 dacă (A) $\geqslant$ (*r*) și CY = 1 dacă (A) < (*r*), pentru numerele cu același semn, sau CY = 1 dacă (A) > (*r*) și CY = 0 dacă (A) $\leqslant$ (*r*), pentru numerele cu semne contrare.

Exemplu. Dacă (A) = 4FH și (H) = 3EH, atunci instrucțiunea SUB H se execută în felul următor:

$$\begin{array}{rl} 4FH = & 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ +(-3EH) = & 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \quad \text{complementul față de 2 al scăzătorului} \\ + & \qquad\qquad\qquad 1 \\ \hline & 1]\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1 = 11H \end{array}$$

valoarea indicatorilor fiind: CY = 0, AC = 1, Z = 0, P = 1 și S = 0.

6. *Exemplu.* Dacă (A) = 0FCH și (D) = 0FH, atunci instrucțiunea ANA D se execută în felul următor:

$$\begin{array}{l} (A) = 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0 = 0FCH \\ (D) = 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1 = 0FH \\ \hline (A) = 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 = 0CH \end{array}$$

iar valoarea indicatorilor este: Z = 0, S = 0, P = 1 și CY = 0.

7. *Exemplu.* Dacă (A) = 5CH și (E) = 78H, atunci instrucțiunea XRA E se execută în felul următor:

$$\begin{array}{l} (A) = 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \\ (E) = 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0 \\ \hline (A) = 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0 = 24H \end{array}$$

iar valoarea indicatorilor este: Z = 0, S = 0 P = 1 și CY = 0.

Instrucțiunea XRA A este utilizată pentru ștergerea acumulatorului și a indicatorilor testabili prin program, Z, S, P și CY.

8. *Exemplu.* Dacă (A) = 33H și (B) = 4FH, atunci instrucțiunea ØRA B se execută în felul următor:

$$\begin{array}{l} (A) = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \\ (B) = 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ \hline (A) = 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 = 7FH \end{array}$$

iar valoarea indicatorilor este: Z = 0, S = 0, P = 0 si CY = 0.

9. Operația de comparare se face prin scăderea conținutului registrului *r* din conținutul acumulatorului, fără a altera conținutul celor două registre, și are ca efect poziționarea indicatorilor de condiții în funcție de rezultatul scăderii (vezi obs. 5).

10. *Constanta D8* este conținută în cel de-al doilea octet al instrucțiunii (fig. 5.3*b*).

11. *Constanta D16/adresa Adr* este conținută în octeții doi, biții de pondere inferioară, și trei, biții de pondere superioară, ai instrucțiunii.

12. Cu ajutorul instrucțiunii LXI SP, D16 se poziționează vîrful stivei de lucru.

13. Cuvîntul indicatorilor de condiții, notat cu litera *F*, are structura:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	0	AC	0	P	1	CY

14. Indicatorul de transport se poziționează, CY = 1, dacă se produce un transport din bitul cel mai semnificativ al rezultatului; în caz contrar este șters.

15. SPHL este utilizată într-un program pentru a poziționa vîrful stivei de lucru.

16. Instrucțiunea DAA convertește conținutul acumulatorului într-un număr binar-zecimal, BCD, și se execută după adunarea a două numere în cod BCD pentru corecția rezultatului. DAA se execută în două faze:

a. Dacă valoarea primilor 4 biți ai acumulatorului, $A_0 \div A_3$, este mai mare decît 9, sau dacă AC = 1, conținutul acumulatorului va fi adunat cu 6, în caz contrar conținutul acumulatorului nu se modifică.

b. Dacă valoarea celor mai semnificativi 4 biți ai acumulatorului, $A_4 \div A_7$, după ce s-a executat faza precedentă, este mai mare decît 9, sau dacă CY = 1, atunci valoarea celor mai semnificativi 4 biți ai acumulatorului va fi adunată cu 6; în caz contrar conținutul acumulatorului nu se schimbă. CY va fi poziționat dacă se produce un transport din bitul cel mai semnificativ, în caz contrar nu va fi afectat.

Exemplu. Dacă (A) = 9BH, CY = 0 și AC = 0, atunci DAA se execută în felul următor:

a. Deoarece valoarea primilor 4 biți ai acumulatorului este mai mare decît 9, conținutul acumulatorului se adună cu 6. Această operație va genera un transport în afara primilor 4 biți, ceea ce va determina poziționarea indicatorului AC, AC = 1

$$\begin{array}{rl} (A) & = 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\ +6 & \qquad\quad 0\ 1\ 1\ 0 \\ \hline (A) & = 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1 = 0A_{16} \end{array}$$

b. Acum valoarea biților 4÷7 ai acumulatorului este mai mare decît 9, ceea ce implică adunarea conținutului acestor biți cu 6. Operația generează un transport în afara bitului cel mai semnificativ, determinînd poziționarea indicatorului CY, CY=1:

$$\begin{array}{rl} (A) & = 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1 \\ +6\times 2^4 & = 0\ 1\ 1\ 0 \\ \hline & 1]\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \end{array}$$

17. Indice L — octetul cel mai puțin semnificativ al unui registru dublu, *rp* sau PC; indice H — octetul cel mai semnificativ al unui registru dublu, *rp* sau PC.

18. Numărătorul de adresă program este incrementat la adresa instrucțiunii următoare, (PC) ← (PC) + 1, și procesorul intră în starea de așteptare pe care nu o poate părăsi decît la apariția unei întreruperi.

19. IN produce transferul în acumulator a unei informații de 8 biți depusă pe magistrala de date de către perifericul cu adresa *Exp*.

20. ØUT permite transferul unei informații de 8 biți din acumulator prin intermediul magistralei de date către perifericul cu adresa *Exp*.

21. *cc* = Z (Zero), NZ (Non Zero), C (Carry), NC (Non Carry), P (Positive), M (Minus), PE (Parity Even), PO (Parity Odd).

5.3.5. PSEUDOINSTRUCȚIUNI

ØRG *Exp*

Definește adresa la care este asamblată instrucțiunea ce urmează după aceasta (fig. 5.5). Numărătorul de adresă-program va fi poziționat în cadrul operației de asamblare la valoarea expresiei din cîmpul operand, *Exp*.

În principiu ORG apare la începutul unui program sau în interiorul lui, dacă se dorește a se modifica adresa secvenței de program ce urmează. În urma asamblării acesteia, asamblorul nu va genera cod-obiect; eticheta este opțională.

```
ASAMBLOR SD-8080 V:3 C.C.A.B. PAG.01

TABELA SIMBOLI
**************
ALFA    0242
BETA    010E
DATA1   010A
DATA2   0112
DATA3   0154
DST     011A
START   0100
TETA    0128
TETA1   0151

0000    *ERORI*

ASAMBLOR SD-8080 V:3 C.C.A.B. PAG.01

0001                    ;
0002                    ; EXEMPLU DE FOLOSIRE A PSEUDOINSTRUCTIUNILOR
0003                    ;
0004                    ALFA  EQU  242H
0005                          ORG  100H
0006 0100 AF            START:XRA  A
0007 0101 32 42 02            STA  ALFA
0008                    ALFA  SET  279H
0009 0104 06 05               MVI  B,5
0010 0106 78                  MOV  A,B
0011 0107 32 79 02            STA  ALFA
0012 010A 23            DATA1:DB   23H,'P',2+7,ALFA
     010B 50
     010C 09
     010D 79
0013 010E 47            BETA: MOV  B,A
0014 010F 21 0A 01            LXI  H,DATA1
0015 0112 45 23         DATA2:DW   2345H,ALFA,$+5,27H*5
     0114 79 02
     0116 1B 01
     0118 C3 00
0016 011A               DST:  DS   7*2
0017 0128 C6 07         TETA: ADI  7
0018                          ORG  150H
0019 0150 47                  MOV  B,A
0020                    ALFA  SET  321H
0021 0151 21 21 03      TETA1:LXI  H,ALFA
0022 0154 54            DATA3:DB   'TEST DB'
     0155 45
     0156 53
     0157 54
     0158 20
     0159 44
     015A 42
0023                          END

0000    *ERORI*
```

Fig. 5.5. Exemplu de utilizare a pseudoinstrucțiunilor.

END

Indică sfîrșitul unui program, fiind delimitatorul final al său. Detectarea sa de către asamblor va determina sfîrșitul unei faze (treceri) a asamblării. În urma asamblării pseudoinstrucțiunii END nu se va genera cod-obiect. Eticheta este opțională, iar atunci cînd este prezentă, asamblorul îi atribuie valoarea curentă a numărătorului de adresă-program.

[*et:*] DB *listă* (Data Byte)

Definește constante de un octet sau un șir de date cu lungimea de un octet. Cîmpul operand poate conține constante, expresii aritmetice și logice reprezentate pe un octet, sau un șir de caractere ASCII incluse între ghilimele simple (fig. 5.5). Aceste mărimi sînt separate cu ajutorul virgulei. În cadrul operației de asamblare valoarea de 8 biți alocată fiecărei mărimi din cîmpul operand, sau fiecărui caracter ASCII dintr-un șir de caractere se memorează la adresa indicată de valoarea indicatorului numărător adresă-program. În cursul asamblării valoarea numărătorului adresă-program va fi incrementată cu numărul de octeți asamblați. Cîmpul etichetă este opțional.

[*et:*] DW *listă* (Data Word)

Definește mărimi cu lungimea de doi octeți. Cîmpul operand, *listă*, poate conține constante, simboluri, expresii a căror valoare este reprezentată pe doi octeți. Aceste mărimi sînt separate cu ajutorul virgulei. Octetul cel mai puțin semnificativ al unei mărimi din cîmpul operand este memorat la adresa indicată de valoarea numărătorului de adresă program; celălalt octet este memorat la adresa următoare (fig. 5.5). În cursul asamblării valoarea numărătorului adresă-program este incrementată cu numărul de octeți asamblați. Cîmpul etichetă este opțional.

[*et:*] DS *Exp* (Data Storage)

Permite rezervarea unor zone de memorie pentru diferite variabile folosite de program. Cîmpul operand, *Exp*, poate fi o constantă, un simbol sau o expresie. Valoarea mărimii din cîmpul operand reprezintă numărul de octeți ce trebuie rezervați. În urma asamblării unei asemenea pseudoinstrucțiuni numărătorul adresă-program va fi incrementat cu valoarea mărimii din cîmpul operand.

Eticheta este necesară pentru a referi mai ușor zona de memorie rezervată.

nume EQU *Exp*

Permite atribuirea de către asamblor a unei valori simbolului din cîmpul etichetă. Un simbol definit cu ajutorul acestei pseudoinstrucțiuni nu mai poate fi definit în altă parte în cadrul aceluiași program. Cîmpul etichetă este separat de cîmpul cod cu unul sau mai multe blancuri. Cîmpul operand, *Exp*, poate fi o constantă, un simbol anterior definit sau o expresie.

nume SET *Exp*

Permite atribuirea de către asamblor a unei valori simbolului din cîmpul etichetă. Spre deosebire de pseudoinstrucțiunea precedentă, SET permite redefinirea unui simbol în cursul aceluiași program ori de cîte ori este necesar. Sintaxa pseudoinstrucțiunii SET este identică cu cea a pseudoinstrucțiunii EQU (fig. 5.5). Aceasta este folosită de către programator în situațiile în care se dorește ca într-un program un simbol să fie refolosit cu valori diferite.

Tabelul 5.1 **Codurile hexazecimale ale instrucțiunilor** *8080*

Operații cu acumulatorul

80 ADD B
81 ADD C
82 ADD D
83 ADD E
84 ADD H
85 ADD L
86 ADD M
87 ADD A
88 ADC B
89 ADC C
8A ADC D
8B ADC E
8C ADC H
8D ADC L
8E ADC M
8F ADC A
90 SUB B
91 SUB C
92 SUB D
93 SUB E
94 SUB H
95 SUB L
96 SUB M
97 SUB A
98 SBB B
99 SBB C
9A SBB D
9B SBB E
9C SBB H
9D SBB L
9E SBB M
9F SBB A
A0 ANA B
A1 ANA C
A2 ANA D
A3 ANA E
A4 ANA H
A5 ANA L
A6 ANA M
A7 ANA A
A8 XRA B
A9 XRA C
AA XRA D
AB XRA E
AC XRA H
AD XRA L
AE XRA M
AF XRA A
B0 ØRA B
B1 ØRA C
B2 ØRA D
B3 ØRA E
B4 ØRA H
B5 ØRA L
B6 ØRA M
B7 ØRA A
B8 CMP B
B9 CMP C
BA CMP D
BB CMP E
BC CMP H
BD CMP L
BE CMP M
BF CMP A

Transferuri

40 MØV B,B
41 MØV B,C
42 MØV B,D
43 MØV B,E
44 MØV B,H
45 MØV B,L
46 MØV B,M
47 MØV B,A
48 MØV C,B
49 MØV C,C
4A MØV C,D
4B MØV C,E
4C MØV C,H
4D MØV C,L
4E MØV C,M
4F MØV C,A
50 MØV D,B
51 MØV D,C
52 MØV D,D
53 MØV D,E
54 MØV D,H
55 MØV D,L
56 MØV D,M
57 MØV D,A
58 MØV E,B
59 MØV E,C
5A MØV E,D
5B MØV E,E
5C MØV E,H
5D MØV E,L
5E MØV E,M
5F MØV E,A
60 MØV H,B
61 MØV H,C
62 MØV H,D
63 MØV H,E
64 MØV H,H
65 MØV H,L
66 MØV H,M
67 MØV H,A
68 MØV L,B
69 MØV L,C
6A MØV L,D
6B MØV L,E
6C MØV L,H
6D MØV L,L
6E MØV L,M
6F MØV L,A
70 MØV M,B
71 MØV M,C
72 MØV M,D
73 MØV M,E
74 MØV M,H
75 MØV M,L
77 MØV M,A
78 MØV A,B
79 MØV A,C
7A MØV A,D
7B MØV A,E
7C MØV A,H
7D MØV A,L
7E MØV A,M
7F MØV A,A

Rotații

07 RLC
0F RRC
17 RAL
1F RAR

Încărcări/memorări

0A LDAX B
1A LDAX D
2A LHLD *Adr*
3A LDA *Adr*
02 STAX B
12 STAX D
22 SHLD *Adr*
32 STA *Adr*

Operații cu stiva

C5 PUSH B
D5 PUSH D
E5 PUSH H
F5 PUSH PSW
C1 PØP B
D1 PØP D
E1 PØP H
F1 PØP PSW
E3 XTHL
F9 SPHL

Speciale

EB XCHG
27 DAA
2F CMA
37 STC
3F CMC
00 NØP
76 HLT
F3 DI
FB EI

Operații imediate cu acumulatorul

C6 ADI
CE ACI
D6 SUI
DE SBI
E6 ANI
EE XRI
F6 ØRI
FE CPI
} *D8*

Incrementări

04 INR B
0C INR C
14 INR D
1C INR E
24 INR H
2C INR L
34 INR M
3C INR A
03 INX B
13 INX D
23 INX H
33 INX SP

Decrementări

05 DCR B
0D DCR C
15 DCR D
1D DCR E
25 DCR H
2D DCR L
35 DCR M
3D DCR A
0B DCX B
1B DCX D
2B DCX H
3B DCX SP

Adunări pe 16 biți

09 DAD B
19 DAD D
29 DAD H
39 DAD SP

Transfer imediat

06 MVI B,
0E MVI C,
16 MVI D,
1E MVI E,
26 MVI H,
2E MVI L,
36 MVI M,
3E MVI A,
} *D8*

Încărcări imediate

01 LXI B,
11 LXI D,
21 LXI H,
31 LXI SP,
} *D16*

Salturi

C3 JMP
C2 JNZ
CA JZ
D2 JNC
DA JC
E2 JPØ
EA JPE
F2 JP
FA JM
E9 PCHL
} *Adr*

Apel subrutine

CD CALL
C4 CNZ
CC CZ
D4 CNC
DC CC
E4 CPØ
EC CPE
F4 CP
FC CM
} *Adr*

Reveniri

C9 RET
C0 RNZ
C8 RZ
D0 RNC
D8 RC
E0 RPØ
E8 RPE
F0 RP
F8 RM

Restart

C7 RST 0
CF RST 1
D7 RST 2
DF RST 3
E7 RST 4
EF RST 5
F7 RST 6
FF RST 7

Intrare/ieșire

D3 ØUT
DB IN
} *D8*

Notații:

Adr = adresă pe 16 biți
D8 = constantă pe 8 biți
D16 = constantă pe 16 biți

5.4. TEHNICI DE PROGRAMARE

5.4.1. TESTAREA INDICATORILOR DE CONDIȚII

Indicatorii de condiții sînt elemente importante ale arhitecturii unui microprocesor, fiind folosiți în majoritatea aplicațiilor ca elemente de decizie. În programarea microprocesoarelor controlul indicatorilor de condiții este esențial pentru a efectua diverse salturi într-un program.

În general, toate operațiile care se execută prin intermediul unității aritmetice-logice, UAL, afectează indicatorii de condiții, în timp ce operațiile care se execută în afara UAL nu-i afectează, cu excepția instrucțiunii PØP PSW. În funcție de valoarea indicatorilor Z, S, P și CY instrucțiunile de salt condiționat implementează mecanismul pentru realizarea programelor cu ramuri multiple. Așa cum s-a prezentat în § 5.3.4 indicatorii de condiții furnizează un număr de 8 condiții de salt diferite.

Uneori este necesară memorarea temporară a indicatorilor pentru a preîntîmpina alterarea lor în diverse secvențe de program, cum sînt:

— subrutine de I/E;

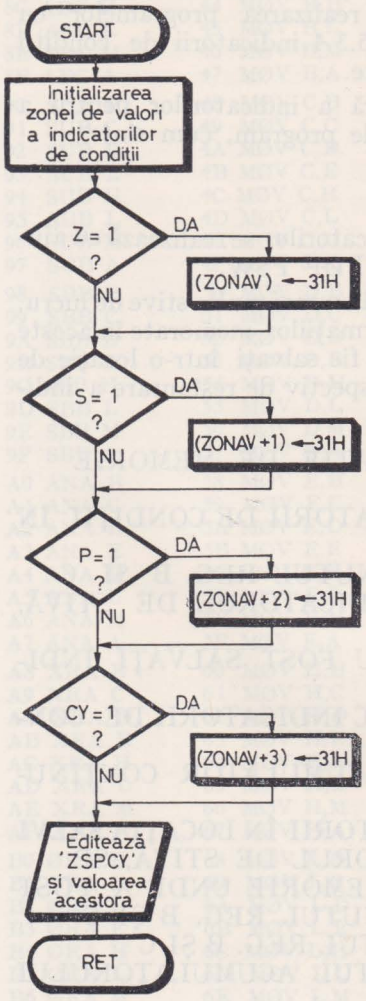
— programe de tratare a întreruperilor etc.

Operația de memorare și restaurare a indicatorilor se realizează cu ajutorul instrucțiunilor PUSH PSW și, respectiv, PØP PSW.

În unele aplicații mai complexe, care operează cu mai multe stive de lucru, fiind mai dificil să se țină o evidență strictă a informațiilor memorate în aceste stive, se recomandă ca indicatorii de condiții să fie salvați într-o locație de memorie special definită. Operația de salvare, respectiv de restaurare a indicatorilor este ilustrată în exemplul următor:

```
SALVI: DS    1      ;REZERVĂ O LOCAȚIE DE MEMORIE
       ----------
       PUSH  PSW    ;SALVEAZĂ INDICATORII DE CONDIȚII ÎN
                    ;STIVĂ
       PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
       INX   SP     ;POZIȚIONEAZĂ INDICATORUL DE STIVĂ,
                    ;SP, PE
       INX   SP     ;LOCAȚIA UNDE AU FOST SALVAȚI INDI-
                    ;CATORII
       PØP   B      ;ÎNCARCĂ ÎN REG. C INDICATORII DE CON-
                    ;DIȚII
       MØV   A,C    ;TRANSFERĂ ÎN ACUMULATOR CONȚINU-
                    ;TUL REG. C
       STA   SALVI  ;SALVEAZĂ INDICATORII ÎN LOCAȚIA SALVI
       PUSH  B      ;REFACE INDICATORUL DE STIVĂ, SP
       DCX   SP     ;LA LOCAȚIA DE MEMORIE UNDE A FOST
       DCX   SP     ;MEMORAT CONȚINUTUL REG. B ȘI C
       PØP   B      ;REFACE CONȚINUTUL REG. B ȘI C
       PØP   PSW    ;REFACE CONȚINUTUL ACUMULATORULUI
```

```
----------          ;SECVENȚĂ DE PROGRAM
----------
LDA   SALVI  ;ÎNCARCĂ ÎN REG. A VALOAREA INDICATO-
             ;RILOR, ANTERIOR SALVATĂ.
PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
MØV   C,A    ;TRANSFERĂ CUVÎNTUL INDICATORILOR
             ;ÎN REG. C
PUSH  B      ;SALVEAZĂ INDICATORII ÎN STIVA DE LU-
             ;CRU
PØP   PSW    ;REFACE VALOAREA INDICATORILOR
PØP   B      ;REFACE CONȚINUTUL REG. B ȘI C
----------          ;SECVENȚĂ DE PROGRAM
```



Atragem atenția că în scrierea secvențelor de salvare, restaurare și interogare a indicatorilor de condiții nu trebuie utilizate instrucțiuni care să-i altereze.

În anumite aplicații este necesar ca valoarea indicatorilor să poată fi listată la consolă sau afișată pe ecranul unui display. În figura 5.6 este prezentată organigrama unui program simplu de testare a tuturor indicatorilor de condiții și de listare a lor la consolă. *Listing*-ul

Fig. 5.6. Algoritm de editare a conținutului indicatorilor de condiții.

acestui program este dat în continuare:

```
ZØNIN: DB   'ZSPCY', 0DH, 0AH
ZØNAV:DS    4              ;ZONA DE MEMORARE A VALORII INDI-
                           ;CATORILOR ÎN ASCII
       PUSH PSW            ;SALVEAZĂ INDICATORII DE CONDIŢII
       MVI  B,4            ;INTRODUCE CODUL ASCII AL CIFREI 0
       LXI  H,ZØNAV+4      ;ÎN ZØNAV
       DCX  H
       MVI  M,30H
       DCR  B              ;DECREMENTEAZĂ CU 1 CONTOR LUN-
                           ;GIME ZØNAV
       JNZ  $-4            ;SALT DACĂ VALOAREA CONTORULUI
                           ;NU ESTE ZERO
       PØP  PSW            ;REFACE INDICATORII DE CONDIŢII
       JNZ  $+5            ;Z=0?
       MVI  M,31H          ;NU, MEMOREAZĂ ÎN ZØNAV VALOA-
                           ;REA LUI Z (Z = 1)
       INX  H              ;INCREMENTEAZĂ CU 1 ADR. ZONEI DE
                           ;MEMORARE
       JP   $+5            ;S=0?
       MVI  M,31H          ;NU, MEMOREAZĂ ÎN ZØNAV+1 VA-
                           ;LOAREA LUI S (S=1)
       INX  H              ;INCREMENTEAZĂ CU 1 ADR. ZONEI DE
                           ;MEMORARE
       JPØ  $+5            ;P=0?
       MVI  M,31H          ;NU, MEMOREAZĂ ÎN ZØNAV+2 VA-
                           ;LOAREA LUI P (P=1)
       INX  H              ;INCREMENTEAZĂ CU 1 ADR. ZONEI DE
                           ;MEMORIE
       JNC  $+5            ;CY = 0?
       MVI  M,31H          ;NU, MEMOREAZĂ ÎN ZØNAV + 3 VA-
                           ;LOAREA LUI CY (CY=1)
;SECVENŢĂ DE EDITARE A VALORII INDICATORILOR ANTERIOR
;MEMORAŢI LA CONSOLĂ
       MVI  B,2
       LXI  H,ZØNIN+5
       CALL TIPMS          ;ÎNTOARCE CARUL ŞI AVANS LINIE
                           ;NOUĂ
       MVI  B,7
       LXI  H,ZØNIN
       CALL TIPMS          ;EDITEAZĂ ZSPCY ŞI CR,LF
       MVI  B,4
       LXI  H,ZØNAV
       CALL TIPMS          ;EDITEAZĂ VALOAREA INDICATORILOR
       MVI  B,2
       LXI  H,ZØNIN+5      ;ÎNTOARCE CARUL ŞI AVANS LA LINIE
                           ;NOUĂ
```

```
TIPMS: MØV   C,M      ;EDITEAZĂ UN MESAJ CONȚINUT
       CALL  CØ       ;ÎNTR-O ZONĂ DE MEMORIE, ADRESATĂ
                      ;INDIRECT CU AJUTORUL CONȚINUTU-
       INX   H        ;LUI REGISTRELOR H, L
       DCR   B        ;ȘI LUNGIME EGALĂ CU CONȚINUTUL
       JNZ   TIPMS    ;REGISTRULUI B
       RET
```

5.4.2. TESTAREA VALORII ANUMITOR BIȚI

În diferite aplicații este necesară testarea valorii anumitor biți dintr-un octet. Această operație se realizează astfel:

— octetul ce conține biții a căror valoare trebuie testată este transferat în acumulator;

— se reține numai biții ce trebuie testați prin mascarea celorlalți, adică prin aducerea la zero a biților ce nu interesează, cu ajutorul unui ȘI LOGIC.

— cu ajutorul instrucțiunilor de comparație, scădere, sau sumă modulo 2, care poziționează indicatorii de condiții, se poate decide asupra valorii acestor biți.

Exemplu

```
VAL: DB   ----------   ;VALOAREA DE REFERINȚĂ IMPUSĂ DE
                       ;PROGRAMATOR
     LDA  ADR          ;ÎNCARCĂ ÎN REG. A OCTETUL CE TREBUIE
                       ;TESTAT
     ANI  MASC         ;REȚINE BIȚII CE TREBUIE TESTAȚI
     CPI  VAL          ;COMPARĂ (A) CU VALOAREA PRESTABI-
                       ;LITĂ
     JZ   ADR1         ;SALT DACĂ (A) = VAL
     JC   ADR2         ;SALT DACĂ (A) ESTE MAI MIC DECÎT VAL
     ----------        ;CONȚINUTUL REG. A ESTE MAI MARE DE-
                       ;CÎT VAL
```

Un alt procedeu de testare a valorii unui bit dintr-un octet constă în încărcarea octetului respectiv în acumulator, deplasarea de un număr corespunzător de ori a conținutului acumulatorului pentru a aduce bitul în cauză în indicatorul de transport, CARRY, și testarea acestuia.

5.4.3. TESTAREA VALORII UNUI OCTET

Valoarea unui octet se poate testa prin compararea acestuia cu diferite valori prestabilite, utilizînd instrucțiunile de comparare, de scădere sau instrucțiunile logice.

Exemplu

```
VAL: DB     ----------    ;VALOAREA DE REFERINȚĂ IMPUSĂ DE
                          ;PROGRAMATOR
     LDA    ADR           ;ÎNCARCĂ ÎN REG. A OCTETUL CE TRE-
                          ;BUIE TESTAT
     CPI    VAL           ;COMPARĂ (A) CU O VALOARE PRESTA-
                          ;BILITĂ
     JZ     ----------    ;(A) = VAL
     JNC    ----------    ;(A) > VAL
     ----------           ;(A) < VAL
```

Instrucțiunile logice sumă modulo 2 și produs logic se folosesc la verificarea egalității între două mărimi.

Exemplu

```
XRI    VAL    ;COMPARĂ IMEDIAT (A) CU VAL
JZ     ADR    ;(A) = VAL
----------    ;(A) ≠ VAL
```

Instrucțiunea cel mai frecvent folosită este cea de comparare, deoarece nu distruge conținutul acumulatorului și furnizează cele mai multe informații despre mărimea chestionată. Faptul că nu distruge conținutul acumulatorului permite o codificare simplificată a testărilor în lanț ale aceleiași mărimi.

5.4.4. TESTAREA CONȚINUTULUI UNUI DUBLU CUVÎNT

Valoarea unui dublu cuvînt poate fi comparată cu o mărime impusă de programator sau cu o mărime rezultată în urma unor prelucrări anterioare. În funcție de scopul urmărit de aplicație, testul poate verifica numai egalitatea celor 2 mărimi, sau pote fi mai complicat.

Exemple

1. Se verifică dacă valoarea conținută în registrele D și E este egală cu valoarea obținută în urma unei prelucrări anterioare, conținută în locațiile de memorie cu adresă ADR+1 și, respectiv, ADR. Pentru a realiza acest lucru este indicată folosirea operației logice sumă modulo 2. Dacă cele două mărimi sînt egale se va poziționa indicatorul Z, Z = 1, iar în caz contrar el va fi șters, Z=0.

```
LDA  ADR      ;ÎNCARCĂ ÎN REG. A OCTETUL MAI PU-
              ;ȚIN SEMNIFICATIV AL MĂRIMII DE
              ;COMPARAT
XRA  E        ;A ← (A) ⊕ (E)
JNZ  ALARM    ;SALT ÎN CAZ DE INEGALITATE
LDA  ADR+1    ;ÎNCARCĂ ÎN REG. A OCTETUL CEL MAI
              ;SEMNIFICATIV AL MĂRIMII DE COM-
              ;PARAT
```

```
        XRA   D        ;COMPARĂ OCTEȚII CEI MAI SEMNIFICA-
                       ;TIVI
        JNZ   ALARM    ;SALT ÎN CAZ DE INEGALITATE
        ----------     ;IEȘIRE PE EGALITATE
ALARM:  ----------     ;CELE 2 MĂRIMI COMPARATE NU SÎNT
                       ;EGALE
```

Compararea unui dublu cuvînt conținut în registrele D și E cu o mărime impusă se poate realiza cu ajutorul următoarei secvențe de program:

```
        LXI   H, VAL   ;ÎNCARCĂ ÎN REG. H ȘI L VALOAREA
                       ;IMPUSĂ
        MØV   A,L      ;ÎNCARCĂ ÎN REG. A CONȚINUTUL REG.L

        XRA   E        ;(A) = (E)?
        JNZ   ALARM    ;NU, SALT LA ADRESA ALARM
        MØV   A,H      ;ÎNCARCĂ ÎN REG. A CONȚINUTUL
                       ;REG. H
        XRA   D        ;(A) = (D)?
        JNZ   ALARM    ;NU, SALT DACĂ Z=0
        ----------     ;IEȘIRE PE EGALITATE
ALARM:  ----------     ;SECVENȚĂ PROGRAM TRATARE INE-
                       ;GALITATE MĂRIMI
```

2. Pentru a obține mai multe informații despre valoarea conținută într-un dublu cuvînt se folosește instrucțiunea de comparație cu o valoare impusă de programator. Pentru aceasta, presupunem că dublul cuvînt, a cărui valoare dorim s-o verificăm, se află în registrele B și C, iar în registrele D și E se încarcă mărimea de referință.

```
        LXI   D,D16    ;(D) ← D16_{15÷8}, (E) ← D16_{7÷0}
        MØV   A,E      ;TRANSFERĂ CONȚINUTUL REG. E ÎN
                       ;ACUMULATOR
        CMP   C        ;COMPARĂ (A) CU (C)
        PUSH  PSW      ;SALVEAZĂ INDICATORII DE CONDIȚII
        MØV   A,D      ;TRANSFERĂ CONȚINUTUL REG. D ÎN
                       ;ACUMULATOR
        CMP   B        ;COMPARĂ (A) CU (B)
        JC    ADR2     ;(B)(C) ESTE MAI MIC DECÎT MĂRIMEA
                       ;DE REFERINȚĂ
        JZ    ADR1     ;(B) = (A)
ADR:    PØP   PSW      ;(B)(C) ESTE MAI MARE DECÎT MĂRIMEA
                       ;DE REFERINȚĂ
        ----------     ;ALTE INSTRUCȚIUNI DIN PROGRAM
ADR1:   PØP   PSW      ;REFACE INDICATORII DE CONDIȚII
        JZ    $+9      ;(B)(C) ESTE EGAL CU MĂRIMEA DE RE-
                       ;FERINȚĂ
        JC    ADR2     ;(B)(C) ESTE MAI MIC DECÎT MĂRIMEA
                       ;DE REFERINȚĂ
```

```
        JMP   ADR+1     ;(B)(C) ESTE MAI MARE DECÎT MĂRIMEA
                        ;DE REFERINȚĂ
        ----------      ;SECVENȚĂ DE PROGRAM CE TRATEAZĂ
                        ;EGALITATEA
                        ;CELOR DOUĂ MĂRIMI COMPARATE
ADR2:   ----------      ;SECVENȚĂ DE PROGRAM CE TRATEAZĂ
                        ;SITUAȚIA ÎN CARE (B)(C) ESTE MAI MIC
        ----------      ;DECÎT MĂRIMEA DE REFERINȚĂ.
```

3. Testarea la zero a valorii unui cuvînt dublu se face utilizînd următoarea secvență de program:

```
        ----------      ;DUBLUL CUVÎNT ESTE ÎNCĂRCAT ÎN
                        ;REG. B și C
        MØV   A,B       ;TRANSFERĂ CONȚINUTUL REG. B ÎN
                        ;ACUMULATOR
        ØRA   C
        JZ    ADR       ;SALT LA INSTRUCȚIUNEA DE LA ADR
                        ;DACĂ (B)(C)=0
        ----------      ;(B)(C) ≠ 0
```

5.4.5. SUBRUTINE

Într-un program o anumită secvență se poate repeta de mai multe ori. Repetarea codificării ei nu este economică din punctul de vedere al utilizării memoriei. Pentru a evita această situație, secvențele care apar de mai multe ori în program se pot scrie o singură dată, și poartă numele de *subrutine*. În scopul de a deveni subrutine, acestora li se adaugă cîteva instrucțiuni suplimentare, pentru a reveni în programul ce le apelează în mod corect, după ce subrutina s-a terminat. Subrutinele sînt apelate de un program numit și *program principal* — ori de cîte ori sînt necesare în execuția acestuia. Transferul comenzii între subrutine și programele principale se numește *legarea subrutinei*.

O subrutină primește un nume și este apelată prin intermediul acestuia; de obicei numele subrutinei este eticheta primei instrucțiuni a ei.

Programatorul apelează o subrutină prin scrierea numelui ei în cîmpul operand al instrucțiunii CALL (vezi § 5.3.4).

Cîteva exemple de operații generale ce se pretează la programarea lor ca subrutine sînt: operațiile de I/E, conversia datelor dintr-un format în altul, manipularea unor liste sau a altor structuri de date, operații matematice pe unul sau mai mulți octeți etc. În afara acestora, pot exista anumite operații sau secvențe de program specifice fiecărei aplicații care sînt convenabil definite ca subrutine.

Căutarea părților care pot fi considerate ca subrutine este recomandabilă datorită avantajelor ce rezultă din folosirea acestui concept:

— economie de memorie și suport pentru programul-sursă (cartele, bandă perforată etc.) prin scurtarea programelor, avînd în vedere faptul că folosirea

subrutinelor elimină necesitatea de scriere repetată a aceluiași grup de instrucțiuni;

— reducerea complexității programelor, simplificarea scrierii, înțelegerii și, mai ales, a punerii la punct a programelor.

Prin utilizarea subrutinelor, programele capătă o structură modulară, fiecare modul, subrutină, reprezentînd o porțiune independentă ce se poate verifica separat, logica de ansamblu a programului devenind în acest fel mai ușor de urmărit și de controlat. În general, în utilizarea subrutinelor apar următoarele probleme:

— legătura dintre programul apelant și subrutină, precum și reîntoarcerea din subrutină în programul apelant;

— transferul de date între programul apelant și subrutină și invers;

— folosirea registrelor interne de către cele 2 unități de program.

În situația în care într-un program se execută o instrucțiune de apel a unei subrutine, de exemplu instrucțiunea CALL, adresa instrucțiunii următoare, conținută în PC, va fi salvată în stiva programului, iar numărătorul de adresă-program, PC, este încărcat cu valoarea operandului instrucțiunii de apel, adresa subrutinei. În continuare se va executa secvența de program de la această adresă, subrutina apelată. Ultima instrucțiune executată într-o subrutină trebuie să fie o instrucțiune de revenire RET, care încarcă în numărătorul de adresă-program, PC, adresa din vîrful stivei; astfel, execuția programului principal continuă cu instrucțiunea următoare (fig. 5.7).

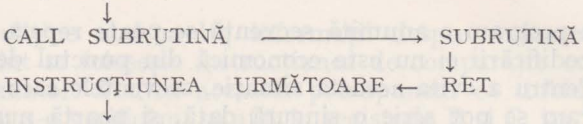


Fig. 5.7 Apelul unei subrutine într-un program

Subrutinele pot apela la rîndul lor alte subrutine, în programe complicate fiind uzuală chemarea de subrutine în lanț pe mai multe niveluri. Numărul subrutinelor dintr-un asemenea lanț este limitat de mărimea zonei de memorie RAM alocată stivei de lucru a programului. În utilizarea unei legături ca cea descrisă aici trebuie să se aibă în vedere restricția ca o subrutină să nu cheme alte subrutine de care, direct sau indirect, este la rîndul ei chemată.

În timpul execuției unei subrutine, în stiva programatorului se pot memora temporar diverse date. Acestea trebuie scoase din stivă înaintea execuției instrucțiunii de revenire, pentru a reface nivelul stivei existent în momentul începerii execuției subrutinei, condiție esențială pentru executarea corectă a revenirii.

Frecvent, o subrutină necesită una sau mai multe date. În cel mai simplu caz, aceste date pot fi transferate prin intermediul registrelor interne ale microprocesorului.

Exemplu:

```
MVI   C,'A'   ;ÎNCARCĂ ÎN REGISTRUL C CODUL
              ;ASCII AL CARACTERULUI A
CALL  CØ      ;TIPĂREȘTE LA CONSOLĂ CARACTERUL A
```

Subrutina CØ va tipări la consolă caracterul al cărui cod ASCII este conținut în registrul C.

Datele necesare unei subrutine pot fi transferate cu ajutorul unei zone de memorie, unde sînt memorate la adrese succesive, iar adresa zonei poate fi comunicată subrutinei prin intermediul unui registru pereche, sau stivei de lucru a programului, ori zona respectivă pote fi apelată direct de subrutina în cauză. În situațiile mai complicate, unde datele necesare într-o subrutină sînt depozitate în zone diferite de memorie, la comunicarea adreselor lor se folosește o altă zonă de memorie în care sînt indicate legăturile cu zonele respective. În practică se pot întîlni multe posibilități de comunicare a datelor unei subrutine, folosind atît registrele interne ale sistemului, cît și memoria afectată aplicației în cauză.

Asemănător, o subrutină poate comunica programului apelant rezultatele prelucrării, atît prin registrele interne, cît și prin zonele de memorie dinainte stabilite sau a căror adresă poate fi transmisă cu ajutorul registrelor duble, stivei etc. De asemenea, rezultatele pot fi comunicate combinat în registre și zone de memorie, cît și prin poziționarea indicatorilor de condiții.

În exemplul următor subrutina ADNR primește în registrele H și L adresa unei liste de date de 3 octeți. Această subrutină adună conținutul primelor două locații de memorie succesive din această zonă și memorează rezultatul acestei operații în cea de-a treia locație. La ieșirea din subrutină conținutul registrelor H și L va indica adresa locației de memorie unde a fost memorat rezultatul operației de adunare. În cadrul exemplului, subrutina ADNR este apelată în două locuri diferite:

```
        LXI   H,ADLIS   ;ÎNCARCĂ HL CU ADRESA LISTEI DE
                        ;PARAMETRI
        CALL  ADNR      ;APELEAZĂ SUBRUTINA DE ADUNARE
RET1:   ----------
        ----------
ADLIS:  DB    6         ;PRIMUL NUMĂR CARE TREBUIE ADUNAT
        DB    8         ;AL DOILEA NUMĂR CE TREBUIE ADUNAT
        DS    1         ;ZONA DE MEMORIE PENTRU REZULTAT
        ----------
        LXI   H,ADLS1   ;ÎNCARCĂ ÎN HL ADRESA ALTEI ZONE
                        ;DE MEMORIE
        CALL  ADNR
RET2:   ----------
        ----------
ADLS1:  DB    10        ;AL DOILEA SET DE DATE
        DB    35        ;UTILIZAT DE SUBRUTINA ADNR ÎN PRO-
                        ;GRAMUL DE MAI SUS
        DS    1
        ----------
        ----------
```

```
ADNR:   MØV   A,M    ;ADUCE ÎN A PRIMUL OPERAND (NUMĂR)
        INX   H      ;INCREMENTEAZĂ ADRESA MEMORIE
        MØV   B,M    ;ADUCE ÎN B AL DOILEA OPERAND (NU-
                     ;MĂR)
        ADD   B      ;ADUNĂ CELE 2 NUMERE
        INX   H      ;INCREMENTEAZĂ ADRESA MEMORIE
        MØV   M,A    ;MEMOREAZĂ REZULTATUL ÎN A TREIA
                     ;LOCAȚIE
        RET          ;REÎNTOARCERE NECONDIȚIONATĂ
```

Zonele de memorie ADLIS și ADLS1 pot apărea oriunde în memoria RAM, fără să afecteze rezultatele produse de subrutina ADNR.

Un exemplu mai complicat este dat în figura 5.13 din § 5.4.8 unde subrutinei de căutare SCAUT îi sînt comunicate următoarele date: în zona de memorie BUF se găsește cuvîntul care trebuie căutat, cheia, în registrele H și L adresa tabelului de date, în BC lungimea tabelului în octeți, în A lungimea unui cuvînt din tabel și în D lungimea valorii asociate unei intrări. După execuția acestei subrutine, ea va comunica programului apelant următoarele date: indicatorul de transport este poziționat, CY = 1, dacă intrarea căutată a fost găsită în tabelul chestionat; în caz contrar CY va fi șters, CY = 0. Poziția de coincidență în cadrul tabelului este memorată de subrutină în zona de memorie IC.

Avînd în vedere că registrele interne constituie de fapt o mică memorie de lucru accesibilă tuturor programelor, se poate întîmpla ca o subrutină să modifice conținutul unor registre în care se află datele necesare pentru execuția corectă a programului principal, apelant, după revenirea din subrutină. Pentru a preveni asemenea modificări nedorite ale conținutului unor registre și indicatorilor de condiții și pentru a nu fi obligat să țină o evidență strictă a utilizării lor de către subrutină, programatorul trebuie să salveze conținutul acelor registre ce nu trebuie alterate și, eventual, al indicatorilor de condiții la intrarea în subrutină. Folosind pentru aceasta stiva de lucru a programului sau o altă zonă de memorie special afectată, programatorul va efectua salvarea registrelor și a indicatorilor ca primă operație la intrarea într-o subrutină, urmînd a le reface apoi conținutul înainte de revenire.

5.4.6. OPERAȚII REPETITIVE, BUCLE

Operațiile repetitive sînt frecvent utilizate într-un proces de control. Ele sînt materializate în programe prin bucle (cicluri) de program controlate cu ajutorul unui contor.

Algoritmul de organizare a unei bucle de program este următorul:

1. Încărcarea contorului, valoarea inițială a lui specifică numărul de iterații ce trebuie realizate.
2. Executarea instrucțiunilor ce compun corpul buclei.
3. Decrementarea și testarea contorului.
4. Oprirea iterației pe condiția de contor nul; în caz contrar se reiau operațiile de la punctul 2.

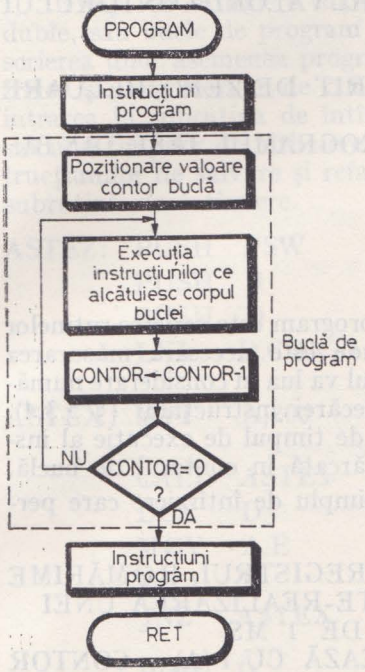


Fig. 5.8. Organigrama unui program general ce folosește o buclă.

În cazul lui 8080 contorul de buclă poate fi reprezentat fie pe un octet, folosind un registru de lucru sau o locație de memorie, fie pe doi octeți cu ajutorul unui registru dublu ori a două locații succesive de memorie.

În primul caz, numărul de bucle ce poate fi executat este cuprins între 1 și 255, în timp ce în al doilea caz el variază între 1 și 65 535.

Organigrama unui program ce folosește o buclă este dată în figura 5.8.

În exemplele 5-4 și 5-5 se prezintă modul de utilizare a contorului de buclă în cele două cazuri.

Exemplul 5-4. Buclă de program cu un contor de iterații mai mic decît 256, încărcat într-un registru de lucru.

```
        MVI  B,D8      ;ÎNCARCĂ ÎN REG. B CONTORUL DE
                       ;ITERAȚII
BUCLA:  --------       ;GRUPUL DE INSTRUCȚIUNI CARE FOR-
                       ;MEAZĂ CORPUL BUCLEI ȘI CARE UR-
        --------       ;MEAZĂ A FI EXECUTATE DE D8 ORI
        DCR  B         ;DECREMENTEAZĂ CU 1 CONTORUL DE
                       ;ITERAȚII
        JNZ  BUCLA     ;TEST LA ZERO AL CONTORULUI DE ITE-
                       ;RAȚII
        --------       ;CONTOR=0, PROGRAMUL IESE DIN BU-
                       ;CLĂ
```

Exemplul 5-5. Buclă de program cu un contor de iterații încărcat într-un registru dublu.

```
        LXI  H,D16     ;ÎNCARCĂ ÎN REG. H ȘI L CONTORUL DE
                       ;ITERAȚII
BUCLA:  --------       ;GRUPUL DE INSTRUCȚIUNI CARE FOR-
                       ;MEAZĂ CORPUL BUCLEI ȘI CARE UR-
        --------       ;MEAZĂ A FI EXECUTATE DE D16 ORI
        DCX  H         ;DECREMENTEAZĂ CU 1 CONTORUL DE
                       ;ITERAȚII
```

```
        MØV   A,L     ;TEST LA ZERO AL VALORII CONTORULUI
                      ;DE
        ØRA   H       ;ITERAȚII
        JNZ   BUCLA   ;CONTOR DIFERIT DE ZERO, RELUARE
                      ;BUCLĂ
        ------        ;CONTOR=0, PROGRAMUL IESE DIN BU-
                      ;CLĂ
```

5.4.6.1. **Rutine de întîrziere (așteptare)**

Un exemplu tipic de utilizare a buclelor de program este scrierea rutinelor de întîrziere, frecvent folosite în aplicațiile unde este necesară măsurarea timpului. Pentru calculul întîrzierii, programatorul va lua în considerare numărul de stări (500 ns/stare) necesar execuției fiecărei instrucțiuni (§ 5.3.4). Mărimea timpului de întîrziere realizat depinde de timpul de execuție al instrucțiunilor folosite în buclă și de valoarea încărcată în contorul de buclă.

În continuare este prezentat un program simplu de întîrziere care permite realizarea unei întîrzieri de (A) × 1 ms.

```
ASTP1: MVI   B,CØNT   ;7/ÎNCARCĂ ÎN REGISTRUL B O MĂRIME
       DCR   B        ;5/CARE PERMITE REALIZAREA UNEI
       JNZ   $—1      ;10/ÎNTÎRZIERI DE 1 MS
       DCR   A        ;5/DECREMENTEAZĂ CU 1 (A) — CONTOR
                      ;NR. MS
       JNZ   ASTP1    ;10/CONTINUĂ PÎNĂ CÎND (A)=0
       RET            ;10/REVENIRE ÎN PROGRAMUL PRINCI-
                      ;PAL
```

În cîmpul comentariu se indică numărul de stări necesare execuției fiecărei instrucțiuni.

Ignorînd timpul de execuție al instrucțiunilor de apel al acestei subrutine, CALL, și de revenire, RET, subrutina de mai sus se execută în:

$$(A) \times (7 + 15 \times C\text{ØNT} + 15) \times 0{,}5 \times 10^{-3} \text{ ms.}$$

Rezultă că pentru a obține o întîrziere de 1 ms, utilizînd subrutina de mai sus, în situația în care valoarea contorului încărcat în registrul A este egală cu 1, în registrul B va trebui încărcată o valoare CØNT dată de relația:

$$22 + 15 \times C\text{ØNT} = 2\,000 \text{ (stări),}$$

de unde

$$C\text{ØNT} = (2\,000 - 22)/15 = 132 = 84_{16}$$

Mărirea timpului de întîrziere se poate realiza cu o subrutină asemănătoare cu cea de mai sus, prin introducerea în interiorul buclei a altor instrucțiuni, după care se recalculează contorul de bucle pentru timpul de așteptare impus.

Timpi de întîrziere mari se pot obține folosind drept contori registre duble, sau bucle de program în interiorul altor bucle (*bucle imbricate*). La scrierea unor asemenea programe de întîrziere distrugerea conținutului registrelor și indicatorilor de condiții poate fi evitată prin salvarea acestora la intrarea în subrutina de întîrziere și refacerea lor la ieșire. În acest caz în calculul timpilor de întîrziere trebuie ținut cont de timpul de execuție al instrucțiunilor de salvare și refacere. În continuare este prezentată o asemenea subrutină de întîrziere.

```
ASTEZ:  PUSH  PSW       ;11/SALVEAZĂ INDICATORII ŞI (A)
        PUSH  B         ;11/SALVEAZĂ CONŢINUTUL REG. B ŞI C
        PUSH  D         ;11/SALVEAZĂ CONŢINUTUL REG. D ŞI E
        LXI   D,CØNT1   ;10/ÎNCARCĂ ÎN REG. DE CONTORUL DE
                        ;ITERAŢII
ASTEX:  MVI   A,200     ;7/ÎNCARCĂ ÎN A UN ALT CONTOR DE
                        ;ITERAŢII
        CALL  ASTE3     ;16/RUTINĂ DE AŞTEPTARE 20 MS
        DCX   D         ;5/DECREMENTARE CU 1/CØNT1
        MØV   A,E       ;5/TEST LA ZERO A VALORII
        ØRA   D         ;4/CONTORULUI CØNT1
        JNZ   ASTEX     ;10/CØNT1=0, RELUARE SECVENŢĂPRO-
                        ;GRAM
        PØP   D         ;10/REFACE CONŢINUTUL REG. D ŞI E
        PØP   B         ;10/REFACE CONŢINUTUL REG. B ŞI C
        PØP   PSW       ;10/REFACE CONŢINUTUL REG. A ŞI IN-
                        ;DICATORILOR
        RET             ;10/REVENIRE ÎN PROGRAMUL PRINCI-
                        ;PAL
ASTE3:  MVI   B,12      ;7/ÎNCARCĂ ÎN REG. B UN CONTOR BU-
                        ;CLĂ
        MØV   C,B       ;5/TRANSFERĂ CONŢINUT REG. B ÎN C
        DCR   C         ;5/DECREMENTEAZĂ CU 1 CONŢINUTUL
                        ;REG. C
        JNZ   $—1       ;10 STĂRI
        DCR   A         ;5/DECREMENTEAZĂ CU 1 CONŢINUTUL
                        ;REG. A
        JNZ   ASTE3+2   ;10/RELUARE BUCLĂ PROGRAM DACĂ
                        ;(A) ≠ 0
        RET             ;10/REVENIRE ÎN PROGRAMUL CHEMĂ-
                        ;TOR
```

Timpul de așteptare realizat de subrutina de întîrziere ASTE3, în situația în care contorul de buclă încărcat în registrul A este egal cu 200, va fi de aproximativ 20 ms, conform relației:

$$(7+(5+(5+10)\times 12+5+10)\times 200+10)/2\,000 = 40\,017/2\,000 = 20 \text{ ms}.$$

Cu ajutorul subrutinei ASTEZ se pot realiza timpi de întîrziere suficient de mari, în funcție de valoarea contorului CØNT1 încărcat în registrul dublu DE. Valoarea contorului CØNT1 se obține din ecuația:

$$11+11+11+10+[CØNT1\times(40\,017+7+16+5+5+4+10)+10\times4=T_{impus}$$

unde: T_{impus} este timpul de așteptare în ms impus de utilizator. Dacă T_{impus} = = 10 s, atunci:

$$CØNT1 = (10\times 1\,000\times 2\,000 - 43 - 40)/40\,064 = 499{,}2.$$

Pentru a realiza timpul de așteptare impus, 10 s, se ia CØNT1 = 500.

5.4.7. TABELE DE SALT

Un program poate conține cîteva subrutine separate ce sînt apelate de programul principal în funcție de valoarea unui parametru (GØ TØ calculat în FØRTRAN). Un procedeu de obținere a unei asemenea secvențe de program care realizează apelul condiționat al unor subrutine este testarea secvențială a fiecărei condiții într-o ordine prestabilită, ca mai jos:

```
CØNDIȚIE = CØNDIȚIE 1 ?
DA,SALT LA SUBRUTINA 1
NU,CØNDIȚIE = CØNDIȚIE 2 ?
DA, SALT LA SUBRUTINA 2
        ⋮
SALT LA SUBRUTINA N
```

Pentru programe cu un număr mare de asemenea condiții acest procedeu nu este eficient, în locul lui utilizîndu-se tehnica *tabelelor de salt.*

Ideea metodei constă în înlocuirea testării secvențiale a fiecărei condiții cu extragerea automată din tabel a adresei de salt corespunzătoare numărului de ordine indicat de parametrul de intrare. De exemplu, se consideră un program ce trebuie să execute o subrutină din cele 8 posibile prin condiționare de poziționarea în 1 a unui bit al acumulatorului:

Salt	la	subrutina	1	dacă	acumulatorul	conține	0	0	0	0	0	0	0	1
„	„		2	„	„	„	0	0	0	0	0	0	1	0
„	„		3	„	„	„	0	0	0	0	0	1	0	0
„	„		4	„	„	„	0	0	0	0	1	0	0	0
„	„		5	„	„	„	0	0	0	1	0	0	0	0
„	„		6	„	„	„	0	0	1	0	0	0	0	0
„	„		7	„	„	„	0	1	0	0	0	0	0	0
„	„		8	„	„	„	1	0	0	0	0	0	0	0

Un program care se execută după această logică este dat în exemplul 5-6. Asemenea program poartă numele de *pseudosubrutină* deoarece este interpretat de programator ca subrutină, apare o singură dată în memorie, iar intrarea într-un astfel de program se face de regulă cu ajutorul unei instrucțiuni de salt (JMP) și nu printr-o instrucțiune de apel a unei subrutine (CALL).

Exemplul 5-6

```
START: LXI   H,ADRUT  ;ÎNCARCĂ ÎN REG. H ŞI L ADRESA TABE-
                      ;LEI DE SALT
TSBIT: RAR            ;TEST BIŢI ACUMULATOR
       JC    INADR    ;SALT DACĂ ULTIMUL BIT A FOST 1
       INX   H        ;(H)(L)=(H)(L)+2, INDICĂ URMĂTOAREA
       INX   H        ;ADRESĂ DE SALT
       JMP   TSBIT    ;RELUARE TEST CONŢINUT ACUMULA-
                      ;TOR
INADR: MØV   E,M      ;ÎNCARCĂ ADRESA SUBRUTINEI
       INX   H        ;SELECTATE ÎN REGISTRELE
       MØV   D,M      ;D şi E;
       XCHG           ;(D) ↔ (H), (E) ↔ (L)
       PCHL           ;TRANSFERĂ CONTROLUL SUBRUTINEI
                      ;SELECTATE
       ----------
ADRUT: DW    RUT1     ;TABELUL DE SALT
       DW    RUT2
       DW    RUT3
       DW    RUT4
       DW    RUT5
       DW    RUT6
       DW    RUT7
       DW    RUT8
```

Un alt exemplu de utilizare a unui tabel de salt este programul ce apelează subrutinele de tratare a pseudoinstrucţiunilor, utilizat de asamblorul microcalculatorului SD-8080.

Simbol (intrare)	Valoare asociată	Structură internă
DB	0 0 0 0 H	44 41 20 00 00
DS	0 0 0 2 H	44 53 20 00 02
DW	0 0 0 4 H	44 57 20 00 04
END	0 0 0 6 H	45 4E 44 00 06
EQU	0 0 0 8 H	45 51 55 00 08
ØRG	0 0 0 A H	4F 52 47 00 0A
SET	0 0 0 C H	53 45 54 00 0C

Fig. 5.9. Tabelul de pseudoinstrucţiuni, TPS.

În cursul operaţiei de asamblare, simbolurile mnemonice ale instrucţiunilor sînt căutate în tabelul pseudoinstrucţiunilor TPS, a cărui structură este indicată în figura 5.9, cu ajutorul rutinei de căutare SCAUT (vezi § 5.4.8, fig. 5.13). În situaţia în care simbolul se găseşte în tabelul TPS, la adresa de memorie IC se va încărca adresa locaţiei de memorie unde este memorat simbolul căutat. În continuare se va extrage din memorie valoarea asociată acestui simbol, care reprezintă adresa relativă a adresei subrutinei de tratare a pseudoinstrucţiunii în tabelul de adrese al rutinelor TRUT1. Adresa absolută a subrutinei se obţine prin adunarea adresei relative cu adresa de început a

tabelei TRUT1, conform următoarei secvențe de program:

```
        LXI     H,TPS     ;ÎNCARCĂ ÎN REG. H ȘI L ADRESA TABE-
                          ;LULUI TPS
        MVI     A,3       ;ÎNCARCĂ ÎN REG. A LUNGIMEA UNUI
                          ;SIMBOL
        MVI     D,2       ;ÎNCARCĂ ÎN REG. D LUNGIMEA VALORII
                          ;ASOCIATE SIMBOLULUI
        LXI     B,7*5     ;ÎNCARCĂ ÎN REG. BC LUNGIMEA TABE-
                          ;LULUI
        CALL    SCAUT     ;CAUTĂ PSEUDOINSTRUCȚIUNEA(INTRA-
                          ;REA) SOLICITATĂ
        JNC     ADRX      ;SALT DACĂ INTRAREA NU A FOST GĂ-
                          ;SITĂ
        LHLD    IC        ;ÎNCARCĂ ÎN REG. H ȘI L ADRESA RELA-
                          ;TIVĂ A INTR.
        INX     H         ;INCREMENTEAZĂ CU 3 ADRESA RELATI-
                          ;VĂ A
        INX     H         ;INTRĂRII ȘI SE OBȚINE ADRESA VALORII
        INX     H         ;ASOCIATE INTRĂRII SOLICITATE ÎN TPS
        XCHG              ;(D) ↔ (H), (E) ↔ (L)
        LDAX    D         ;ÎNCARCĂ ÎN REGISTRELE B ȘI C ADRESA
        MØV     B,A       ;RELATIVĂ A ADRESEI SUBRUTINEI DE
        INX     D         ;TRATARE A PSEUDOINSTRUCȚIUNII
        LDAX    D         ;SOLICITATE ÎN TABELA DE SUBRUTINE
        MØV     C,A       ;TRUT1
        LXI     H,TRUT1   ;ÎNCARCĂ ÎN REG. H ȘI L ADRESA TABE-
                          ;LULUI DE SUBRUTINE
        DAD     B         ;ÎN REG. H ȘI L SE OBȚINE ADRESA
                          ;SUBRUTINEI
        PCHL              ;TRANSFERĂ CONTROLUL RUTINEI SE-
                          ;LECTATE
ADRX:   ---------         ;INTRAREA CĂUTATĂ NU A FOST GĂSITĂ
TRUT1:  DW      RDB1      ;TABELUL CU ADRESELE SUBRUTINELOR
        DW      RDS1      ;DE TRATARE A PSEUDOINSTRUCȚIU-
                          ;NILOR
        DW      RDW1      ;RECUNOSCUTE DE ASAMBLORUL
        DW      REND1     ;MICROCALCULATORULUI SD-8080
        DW      REQU1
        DW      RSET1
TPS:    DB      'DB',0,0
        DB      'DS',0,2
        DB      'DW',0,4
        DB      'END',0,6
        DB      'EQU',0,8
        DB      'ØRG',0,10
        DB      'SET',0,12
LTPS    EQU     $-TPS     ;LUNGIMEA TABELULUI TPS, ÎN OCTEȚI
```

5.4.8. TABELE DE DATE

Majoritatea aplicațiilor cu microprocesoare, inclusiv cu 8080, utilizează *structuri de date* de diferite categorii. În cadrul unei structuri, datele sînt organizate conform unui procedeu stabilit de programator. Modul de organizare a datelor influențează direct caracteristicile de exploatare ale structurii. Acestea se referă în principal la două aspecte oarecum contradictorii: ușurința programării operației de căutare a unui element al structurii și, respectiv micșorarea timpului mediu de căutare. O altă operație care prezintă interes în cadrul acestui paragraf este adăugarea și extragerea de elemente în/din cadrul unei structuri de date alese. Se disting mai multe tipuri de structuri:

— liniare;
— ramificate (de tip arbore);
— buclate.

Aceste structuri sînt fundamentale; orice altă structură poate fi reprezentată cu ajutorul acestora.

Structura liniară este o structură în care datele ce intră în alcătuirea ei respectă o relație de ordine, iar accesul la una din informațiile (datele) acestei structuri se face prin parcurgerea unei succesiuni de informații.

Structura ramificată (de tip arbore) este caracteristică tuturor structurilor ierarhice, fiind compusă din noduri și ramuri. Ramurile sînt structuri liniare, iar nodurile — puncte comune mai multor structuri liniare. În cazul acesteia accesul la o anumită informație se face mai ușor decît în cazul structurii liniare.

Structură buclată reprezintă o structură de date în care există cel puțin o cale de închidere, adică ultimul element al structurii indică pe primul element al ei ca element succesor.

Problema alegerii unei structuri de date este dezvoltată în cadrul altor lucrări [6, 7, 8]. În lucrarea de față ne vom preocupa numai de cele mai simple structuri de date — tabelele.

Tabelele sînt structuri liniare apărînd ca liste secvențiale de elemente, numite de obicei *articole* sau *intrări*, de lungime fixă sau variabilă.

Cel mai simplu tabel de date este acela în care fiecare intrare are lungimea fixă, tabelul fiind complet definit prin adresa primei intrări, lungimea în octeți a fiecărui articol și adresa ultimei intrări, sau numărul de articole conținut în tabel. În cazul unor articole de lungime variabilă, fiecăruia îi va fi asociată o informație privind lungimea sa. Un astfel de tabel va fi definit prin adresa primei intrări, iar fiecare intrare conține în primul octet lungimea sa. Sfîrșitul tabelului este marcat de un articol special, care poate fi o informație nulă — lungimea zero a articolului.

Operațiile care interesează a fi executate într-un tabel de date sînt cele de identificare (căutare), anexare și eliminare a unui articol (intrări).

Consultarea unui tabel de date se poate face în două moduri:

— fiind dat numărul de ordine al intrării se cere să se afle conținutul acesteia;

— fiind dată informația aferentă intrării, cheia de căutare, se cere să se afle dacă această informație este prezentă în tabel; în caz afirmativ se va preciza numărul afectat intrării cu acest conținut, sau adresa acesteia.

În tabelele de date cu intrări de lungime fixă, datele pot fi ordonate sau neordonate.

Cel mai simplu procedeu de căutare a unei intrări într-un tabel de date este *căutarea liniară sau secvențială*. Acest procedeu este folosit în special pentru căutarea unei intrări într-un tabel cu intrări de lungime fixă sau variabilă și constă în compararea exhaustivă a intrării cu fiecare intrare din tabel. Reîntoarcerea din acest program de căutare nu se face pînă cînd nu se stabilește fie coincidența dintre intrarea (cheia) căutată și o intrare a tabelului, fie lipsa din tabel a cheii respective.

Datorită simplității sale, căutarea secvențială este indicată pentru tabelele în format fix de lungime redusă, ea fiind foarte lentă în cazul tabelelor mari. Timpul mediu de căutare liniară a unei intrări într-un tabel cu N intrări este egal cu $(N/2 \times T_s)$, unde T_s este timpul necesar comparării unei intrări.

Pentru a scurta timpul de căutare, în majoritatea aplicațiilor tabelele de date cu format fix folosite se ordonează crescător sau descrescător în ordinea valorilor atribuite conținutului intrărilor; acest procedeu este condiționat de existența unei relații de ordine.

Exemple de tabele de date:

— tabelul de simboluri obținut în urma operației de asamblare, trecerea întîia;

— tabelul codurilor mnemonice ale instrucțiunilor microprocesorului 8080, folosit de asamblor;

— tabelul codurilor mnemonice ale pseudoinstrucțiunilor (fig. 5.9);

— tabelul de corespondențe EBCDIC-ASCII ale tuturor caracterelor unei tastaturi de consolă.

Intrările în tabelul de simboluri obținut în urma operației de asamblare sînt simbolurile întîlnite în cîmpul-etichetă al unui program. Aceste intrări sînt ordonate aritmetic crescător după valoarea corespunzătoare codului ASCII al caracterelor etichetei (fig. 5.5). Valorile asociate acestor simboluri nu sînt luate în considerare la ordonarea tabelului. Asemănător tabelului de simboluri, și celelalte tabele indicate mai sus sînt ordonate crescător după valoarea fiecărei intrări în tabel, mărimea asociată acesteia nefiind luată în considerare. Într-un astfel de tabel simbolurile (intrările) care nu au maximum de caractere vor fi completate cu zerouri sau blancuri, în funcție de convențiile stabilite de programator. Problema căutării unei intrări într-un astfel de tabel este aceea a găsirii coincidenței dintre cuvîntul de intrare, cheia, și unul din articolele tabelului, precum și furnizarea unor informații cu privire la poziția acestuia în tabel și valoarea asociată lui.

Localizarea unei intrări într-un tabel, ordonat crescător sau descrescător, după valoarea (mărimea) intrărilor sale, se face prin compararea numerică a cuvîntului de intrare cu articolul de la mijlocul tabelului; cuvîntul de intrare poate fi egal, mai mare sau mai mic decît acesta, rezultatul interpretîndu-se după cum urmează:

1. dacă este egal, înseamnă că intrarea căutată a fost găsită și operația se termină;

2. dacă este mai mic, înseamnă că intrarea căutată poate fi în prima jumătate a tabelului;

3. dacă este mai mare, înseamnă că intrarea căutată poate fi în a doua jumătate a tabelului.

În continuare, operația se repetă în jumătatea de tabel selectată prin comparația precedentă. Astfel, intrarea va fi localizată într-o pătrime a tabelului, apoi în pasul următor — într-o optime a tabelului, ș.a.m.d.

Operația de căutare se va termina cu condiția *articol negăsit* atunci cînd lungimea ultimului subtabel cercetat va fi mai mică sau egală cu numărul de octeți afectați unei intrări. Această metodă este cunoscută sub numele de *căutare binară* sau *căutare logaritmică*. Utilizînd această metodă o intrare este localizată într-un tabel cu N articole în maximum $\log_2 N$ căutări.

Dacă T_s este timpul necesar executării unui pas în căutarea liniară și T_1 — timpul necesar executării unui pas în căutarea logaritmică, atunci timpul maxim pentru localizarea unei intrări în fiecare din cele două metode va fi dat de:

$$T(\text{lin}) = T_s \cdot N \tag{5.1}$$

$$T(\log) = T_1 \cdot \log_2 N \tag{5.2}$$

Reprezentarea grafică a celor două relații, considerînd în ordonată timpul T și în abscisă numărul de intrări N, scoate în evidență domeniul de utilitate al fiecărei metode (fig. 5.10). Deoarece un pas în căutarea logaritmică într-un tabel de date ordonat este mai complicat decît un pas similar în căutarea secvențială, adică T_1 este mult mai mare decît T_s, pentru tabelele de date cu un număr mic de intrări se indică folosirea unui program de căutare secvențială. Punctul de intersecție a celor două curbe depinde de viteza de lucru și de setul de instrucțiuni al microcalculatorului și variază între 5 și 20 de intrări în funcție de tipul acestuia.

Diferența dintre cele două tipuri de căutări se poate observa și din statistica prezentată în tabelul 5.2.

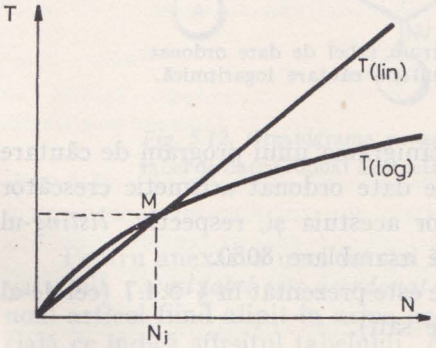


Fig. 5.10. Curbele timpului de căutare într-un tabel de date în funcție de lungimea tabelului

Tabelul 5.2. **Situație statistică privind localizarea unei intrări într-un tabel**

Număr de intrări în tabel N	Căutarea secvențială Nr. mediu de căutări N/2	Căutarea logaritmică Nr. maxim de căutări $\log_2 N$
32	16	5
1 024	512	10
32 678	16 384	15

Un exemplu de căutare logaritmică a unei intrări într-un tabel de date ordonat aritmetic crescător este dat în figura 5.11. Se consideră un tabel de date cu 15 intrări în care valoarea aritmetică a fiecărei intrări o constituie cuvîntul de 16 biți format prin concatenarea codurilor ASCII ale celor două caractere alfanumerice corespunzînd intrării. Ordonarea alfabetică a tabelului va corespunde în acest caz ordonării aritmetice crescătoare datorită ordinii numerice crescătoare a reprezentărilor ASCII ale caracterelor. Se dorește localizarea intrării ce are valoarea corespunzătoare simbolului IF, adică a întregului binar ce corespunde reprezentării interne a caracterelor acestuia. Primul pas al operației de căutare îl constituie compararea mărimii corespunzătoare simbolului IF cu intrarea de la mijlocul tabelului, LØ, de unde rezultă că IF se găsește în prima jumătate. În cel de-al doilea pas, IF este comparat cu FU, intrarea de la mijlocul subtabelului selecționat, și se constată că este mai mare decît acesta. Următorul pas al operației de căutare se va executa asemănător, în a doua parte a subtabelului chestionat anterior.

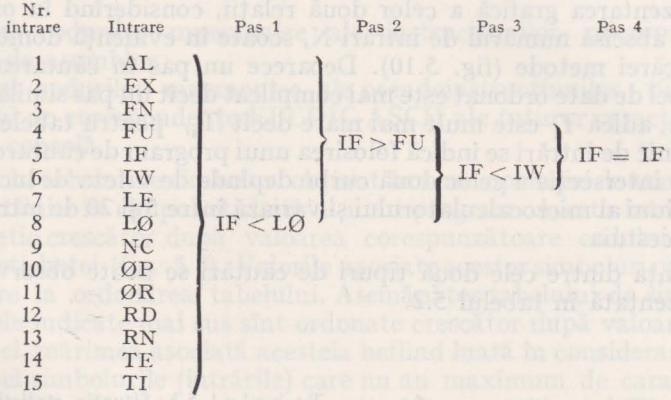


Fig. 5.11 Localizarea unei intrări într-un tabel de date ordonat aritmetic crescător folosind programul de căutare logaritmică.

În figurile 5.12 și 5.13 se prezintă organigrama unui program de căutare logaritmică a unei intrări într-un tabel de date ordonat aritmetic crescător (sau descrescător) după mărimea intrărilor acestuia și, respectiv, *listing*-ul programului corespunzător în limbajul de asamblare 8080.

Un mod de utilizare a acestei subrutine este prezentat în § 5.4.7 (cel de-al doilea exemplu de utilizare a tabelelor de salt).

Anexarea și *eliminarea* unei intrări într-un tabel de date sînt operații ce-i modifică conținutul. Modul de execuție al celor două operații depinde de tipul tabelului — în format variabil și în format fix, ordonat sau neordonat.

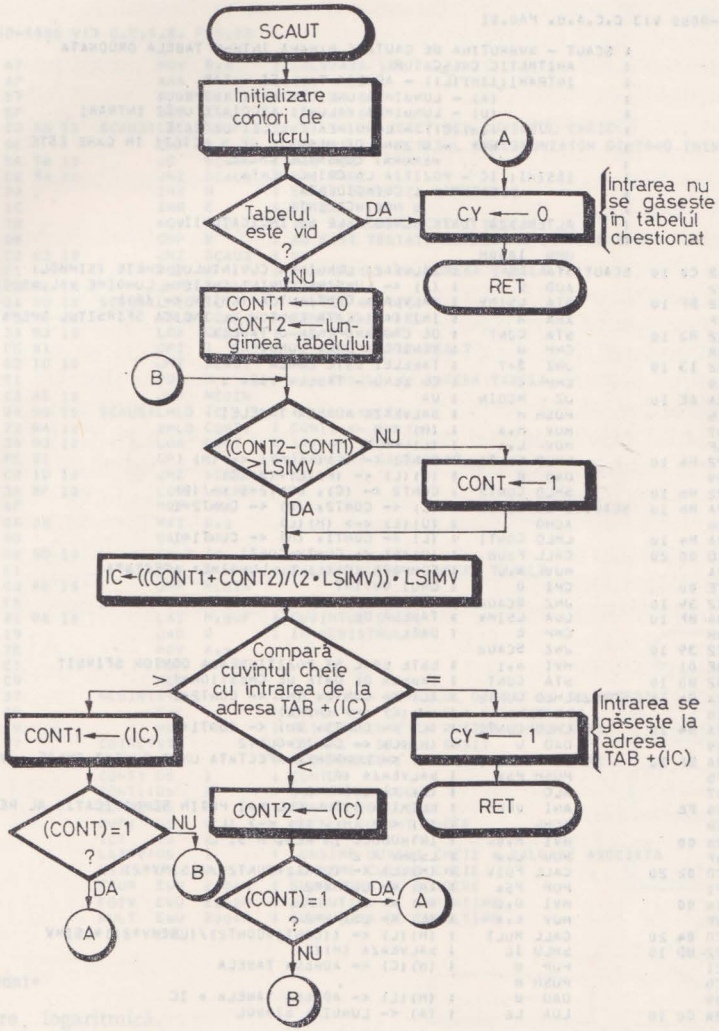


Fig. 5.12. Organigrama programului de căutare logaritmică într-un tabel de date ordonat aritmetic crescător după mărimea intrărilor sale

Pentru anexarea unei intrări într-un tabel de date cu intrări de lungime variabilă *(acest tabel este neordonat)* se caută adresa ultimei intrări a tabelului, noul articol fiind alipit în urma acesteia. În final este memorată intrarea specială ce indică sfîrșitul tabelului. Anexarea unui articol într-un tabel neordonat cu intrări de lungime fixă constă în calculul adresei primei intrări libere în tabel, urmat de memorarea intrării în cauză. În final, se reactualizează parametrii tabelului de date afectați de această operație.

```
ASAMBLOR SD-8080 V:3 C.C.A.B. PAG.01

0001                    ; SCAUT - SUBRUTINA DE CAUTARE BINARA INTR-O TABELA ORDONATA
0002                    ;     ARITMETIC CRESCATOR
0003                    ;     INTRARI:((H)(L)) - ADRESA TABELEI - TAB
0004                    ;              (A) - LUNGIMEA UNEI INTRARI (SIMBOL)
0005                    ;              (D) - LUNGIMEA VALORII ASOCIATE UNEI INTRARI
0006                    ;              ((B)(C)) - LUNGIMEA TABELEI DE DATE
0007                    ;              BUF  - O ZONA DE MEMORIE DE 5 OCTETI IN CARE ESTE
0008                    ;                     MEMORAT CUVINTUL CHEIE
0009                    ;     IESIRI: IC - POZITIA LA COINCIDENTA
0010                    ;             CARRY = 1 COINCIDENTA
0011                    ;                     0 NCOINCIDENTA
0012                    ;     ALTEREAZA TOATE REGISTRELE SI INDICATORII
0013                    ;
0014                            ORG  1000H
0015 1000 32 C0 10      SCAUT:STA  LS       ; SALVEAZA LUNGIMEA CUVINTULUI CHEIE (SIMBOL)
0016 1003 82                    ADD  D      ; (A) <- LUNGIME CUVINT CHEIE + LUNGIME VAL.ASOCIATA
0017 1004 32 BF 10              STA  LSIMV  ; SALVEAZA CONTINUT A (LSIMV <- (A))
0018 1007 AF                    XRA  A      ; INITIALIZEAZA CONTOR CE INDICA SFIRSITUL OPERATIEI
0019 1008 32 B3 10              STA  CONT   ; DE CAUTARE INTR-O TABELA
0020 100B B8                    CMP  B      ; VERIFICA DACA LUNGIMEA
0021 100C C2 13 10              JNZ  $+7    ; TABELEI ESTE EGALA
0022 100F B9                    CMP  C      ; CU ZERO - TABELA VIDA ?
0023 1010 CA AE 10              JZ   NCOIN  ; DA
0024 1013 E5                    PUSH H      ; SALVEAZA ADRESA TABELEI
0025 1014 67                    MOV  H,A    ; (H) <-0
0026 1015 6F                    MOV  L,A    ; (L) <- 0
0027 1016 22 B4 10              SHLD CONT1  ; CONT1 <- (L), CONT1 <- (H)
0028 1019 09                    DAD  B      ; (H)(L) <- (H)(L)+(B)(C)
0029 101A 22 B6 10              SHLD CONT2  ; CONT2 <- (C), CONT2+1 <- (B)
0030 101D 2A B6 10      SCAU1:LHLD CONT2    ; (L) <- CONT2, (H) <- CONT2+1
0031 1020 EB                    XCHG        ; (D)(E) <-> (H)(L)
0032 1021 2A B4 10              LHLD CONT1  ; (L) <- CONT1, (H) <- CONT1+1
0033 1024 CD 00 20              CALL FSUB   ; (D)(E) <- CONT2-CONT1
0034 1027 7A                    MOV  A,D    ; COMPARA (D)(E) CU LUNGIMEA AFECTATA
0035 1028 FE 00                 CPI  0      ; UNUI CUVINT
0036 102A C2 39 10              JNZ  SCAU2  ; CHEIEI IN
0037 102D 3A BF 10              LDA  LSIMV  ; TABELA DE
0038 1030 BB                    CMP  E      ; DATE
0039 1031 C2 39 10              JNZ  SCAU2
0040 1034 3E 01                 MVI  A,1    ; ESTE EGAL SI POZITIONEAZA CONTOR SFIRSIT
0041 1036 32 B3 10              STA  CONT   ; TABELA DE DATE DE CHESTIONAT
0042 1039 2A B6 10      SCAU2:LHLD CONT2    ; (L) <- CONT2, (H) <- CONT2+1
0043 103C EB                    XCHG        ; (D)(E) <-> (H)(L)
0044 103D 2A B4 10              LHLD CONT1  ; (L) <- CONT1, (H) <- CONT1+1
0045 1040 19                    DAD  D      ; (H)(L) <- CONT1+CONT2
0046 1041 3A BF 10              LDA  LSIMV  ; (A) <- LUNGIMEA AFECTATA UNUI CUVINT CHEIE + VALOARE
0047 1044 F5                    PUSH PSW    ; SALVEAZA (A)
0048 1045 07                    RLC         ; (A) <- LSIMV*2
0049 1046 E6 FE                 ANI  0FEH   ; ELIMINA BITUL CEL MAI PUTIN SEMNIFICATIV AL REG.A
0050 1048 EB                    XCHG        ; (D) <-> (H), (E) <-> (L)
0051 1049 26 00                 MVI  H,0    ; INTRODUCE IN REG. H SI L
0052 104B 6F                    MOV  L,A    ; LSIMV * 2
0053 104C CD 02 20              CALL FDIV   ; (H)(L) <- (CONT1+CONT2)/(LSIMV*2)
0054 104F F1                    POP  PSW    ; (A) <- LSIMV*2
0055 1050 16 00                 MVI  D,0    ; (D) <- 0
0056 1052 5F                    MOV  E,A    ; (E) <- LSIMV*2
0057 1053 CD 04 20              CALL MULT   ; (H)(L) <- ((CONT1+CONT2)/(LSIMV*2))*LSIMV
0058 1056 22 BD 10              SHLD IC     ; SALVEAZA (H)(L)
0059 1059 C1                    POP  B      ; (B)(C) <- ADRESA TABELA
0060 105A C5                    PUSH B
0061 105B 09                    DAD  B      ; (H)(L) <- ADRESA TABELA + IC
0062 105C 3A C0 10              LDA  LS     ; (A) <- LUNGIME SIMBOL
```

Fig. 5.13. Listing-ul programului

Se presupune că zonele de memorie afectate tabelelor de date sînt suficient de mari pentru a nu se produce o depășire a zonelor de memorie prin introducerea unui nou articol. În caz contrar, este necesar ca înaintea anexării unei noi intrări să se verifice dacă există spațiu suficient în zona de memorie afectată tabelului respectiv pentru o nouă intrare.

Anexarea unei intrări într-un tabel ordonat cu intrări de lungime fixă necesită următoarele etape:

— se verifică dacă există spațiu suficient în tabel pentru inserarea unei intrări noi;

— se caută adresa unde urmează să se introducă noua intrare;

```
ASAMBLOR SD-8080 V:3 C.C.A.B. PAG.02

0063 105F 47            MOV  B,A    ; SALVEAZA (A)
0064 1060 AF            XRA  A      ; (A) <- 0
0065 1061 57            MOV  D,A    ; (D) <- 0
0066 1062 5F            MOV  E,A    ; (E) <- 0
0067 1063 CD A6 10  SCAU3:CALL SBUF ; (A) <- UN OCTET DIN CUVINTUL CHEIE
0068 1066 BE            CMP  M      ; COMARA (A) OCTETUL CORESPUNZATOR DINTR-O INTRARE A TABELEI
0069 1067 DA 78 10      JC   SCAU4  ; MAI MIC
0070 106A C2 8A 10      JNZ  SCAU5  ; MAI MARE
0071 106D 23            INX  H      ; EGAL, VERIFICA URMATORII
0072 106E 1C            INR  E      ; OCTETI
0073 106F 7B            MOV  A,E    ; (A) <- NUMAR OCTETI TESTATI
0074 1070 B8            CMP  B      ; AU FOST TESTATA COMPLET O INTRARE ?
0075 1071 C2 63 10      JNZ  SCAU3  ; NU
0076 1074 E1            POP  H      ; DA, (H)(L) <- ADRESA TABELA
0077 1075 C3 B1 10      JMP  COINC  ; CY <- 1
0078 1078 2A BD 10  SCAU4:LHLD IC   ; (H)(L) <- (IC)
0079 1078 22 B6 10      SHLD CONT2  ; CONT2 <- (H)(L)
0080 107E 3A B3 10      LDA  CONT   ; (A) <- (CONT)
0081 1081 FE 01         CPI  1      ; TERMINAT CAUTAREA ?
0082 1083 C2 1D 10      JNZ  SCAU1  ; NU
0083 1086 E1            POP  H      ; DA, (H)(L) <- ADRESA TABELA
0084 1087 C3 AE 10      JMP  NCOIN
0085 108A 2A BD 10  SCAU5:LHLD IC   ; (H)(L) <- IC
0086 108D 22 B4 10      SHLD CONT1  ; CONT1 <- (H)(L)
0087 1090 3A B3 10      LDA  CONT   ; (A) <- (CONT)
0088 1093 FE 01         CPI  1      ; TERMINAT CAUTAREA?
0089 1095 C2 1D 10      JNZ  SCAU1  ; NU
0090 1098 3A BF 10      LDA  LSIMV
0091 109B 4F            MOV  C,A
0092 109C 06 00         MVI  B,0
0093 109E 09            DAD  B
0094 109F 22 BD 10      SHLD IC
0095 10A2 E1            POP  H      ; DA, (H)(L) <- ADRESA TABELA
0096 10A3 C3 AE 10      JMP  NCOIN
0097 10A6 E5        SBUF: PUSH H    ; INCARCA UN OCTET DIN
0098 10A7 21 B8 10      LXI  H,BUF  ; CUVINTUL CHEIE
0099 10AA 19            DAD  D      ; IN REGISTRUL A
0100 10AB 7E            MOV  A,M
0101 10AC E1            POP  H
0102 10AD C9            RET
0103 10AE 37        NCOIN:STC       ; CY <- 1 - SIMBOLUL CAUTAT NU SE GASESTE
0104 10AF 3F            CMC         ; CY <- 0   IN TABELA CHESTIONATA
0105 10B0 C9            RET         ; REINTOARCERE IN RUTINA CHEMATOARE
0106 10B1 37        COINC:STC       ; CY <-1 - SIMBOL GASIT
0107 10B2 C9            RET         ; REINTOARCERE IN RUTINA CHEMATOARE
0108 10B3           CONT: DS  1     ; CONTOR
0109 10B4           CONT1:DS  2     ; CONTOR DE LUCRU
0110 10B6           CONT2:DS  2     ; CONTOR DE LUCRU
0111 10B8           BUF:  DS  5     ; BUFFER CUVINT CHEIE
0112 10BD           IC:   DS  2     ;
0113 10BF           LSIMV:DS  1     ; LUNGIME CUVINT CHEIE + VALOARE ASOCIATA
0114 10C0           LS:   DS  1     ; LUNGIME CUVINT CHEIE
0115                FSUB  EQU 2000H ; SUBRUTINA DE SCADERE
0116                FDIV  EQU 2002H ; SUBRUTINA DE IMPARTIRE
0117                MULT  EQU 2004H ; SUBRUTINA DE INMULTIRE
0118                      END


0000   *ERORI*
```

de căutare logaritmică.

— următoarele intrări, începînd cu cea de la adresa anterior calculată, sînt deplasate în jos cu un număr de locații egal cu cel afectat unei intrări;

— se memorează intrarea dorită în adresa anterior calculată;

— se reactualizează parametrii tabelului.

Eliminarea unei intrări dintr-un tabel de date presupune următoarele etape:

— calculul adresei intrării în tabel;

— transferul articolului într-o zonă de memorie, buffer, pentru o eventuală prelucrare;

— compactarea tabelului;

— reactualizarea parametrilor tabelului.

În majoritatea aplicațiilor, tabelele de date în format fix sînt destul de mari. Localizarea unei intrări, în timp cît mai scurt, într-un astfel de tabel, necesită folosirea unui program de căutare logaritmică, ceea ce presupune ordonarea prealabilă a tabelului.

Tabelele cu un număr mare de intrări sînt greu de ordonat manual, ceea ce impune folosirea unui program de ordonare.

Organizarea datelor în ordine crescătoare sau descrescătoare, plecînd de la un tabel neordonat, poartă numele de *sortare*.

Un algoritm simplu de ordonare a unui tabel cu intrări de lungime fixă este *sortarea prin inversare*. În organigrama din figura 5.14 este prezentat algoritmul de sortare prin inversare în ordine crescătoare a valorii intrărilor. Algoritmul presupune parcurgerea într-un sens a tabelului de date, cu NRI intrări, și inversarea concomitentă a perechilor de intrări care nu respectă relația de ordine impusă. Operația de sortare se desfășoară iterativ pînă cînd tabelul va fi ordonat, adică pînă atunci cînd după parcurgerea completă a tabelului nu mai este necesară execuția nici unei inversări, CPRM=0.

Subrutina care realizează sortarea datelor dintr-un tabel de date în conformitate cu algoritmul prezentat în figura 5.14, SØRTI, este prezentată în continuare. Datele de intrare necesare acestei subrutine sînt comunicate cu ajutorul următoarelor registre: H și L — adresa tabelului de date, B — numărul de intrări în tabel — NRI, D — numărul de octeți afectați unei intrări și E — numărul de octeți afectați valorii asociate unei intrări.

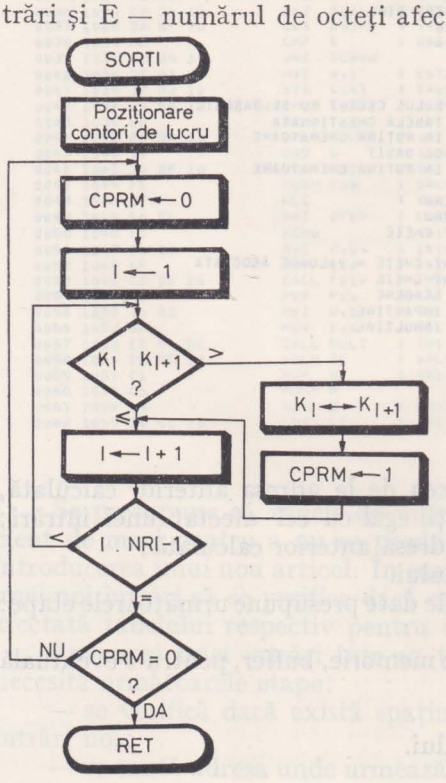


Fig. 5.14. Algoritmul de sortare a intrărilor dintr-un tabel de date în ordine crescătoare, utilizînd metoda inversării.

În cadrul operației de sortare se va lua în considerare numai valoarea unei intrări, de lungime egală cu conținutul registrului D. Tabelul de date ordonat se va obține în locul tabelului inițial, adresa acestuia este indicată de conținutul registrului pereche HL. Subrutina alterează conținutul registrelor A, C, D și E, pecum și al indicatorilor de condiții.

```
SØRTI: PUSH   H       ;SALVEAZĂ CONȚINUTUL REG. H ȘI L —
                      ;ADR. TABELĂ
       MØV    A,D     ;TRANSFERĂ CONȚINUTUL REG. D ÎN
                      ;ACUMULATOR
       STA    LVALI   ;SALVEAZĂ LUNGIMEA VALORII UNEI
                      ;INTRĂRI
       MØV    A,D     ;CALCULEAZĂ NUMĂRUL DE OCTEȚI
                      ;AFECTAȚI
       ADD    E       ;UNEI INTRĂRI ÎN TABELUL DE DATE
       STA    LINTR   ;SALVEAZĂ LUNGIMEA UNEI INTRĂRI
       MØV    C,A     ;ÎNCARCĂ ÎN REG. C LUNGIMEA UNEI
                      ;INTRĂRI
       PUSH   B       ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
SØRT4: XRA    A       ;INTRODUCE ZERO ÎN ACUMULATOR
       STA    CPRM    ;INIȚIALIZEAZĂ CONTOR INVERSĂRI
       INR    A       ;INIȚIALIZEAZĂ CONTOR INTRĂRI ÎN
       STA    I       ;TABELUL DE DATE — I=1
       MØV    D,H     ;TRANSFERĂ ADRESA PRIMEI INTRĂRI ÎN
       MØV    E,L     ;TABELUL DE DATE ÎN REG. D ȘI E
       MVI    B,0     ;CALCULEAZĂ ADRESA CELEI DE-A DOUA
       DAD    B;      ;INTRĂRI ÎN TABEL, K2
       LDA    LVALI   ;ÎNCARCĂ ÎN REG. B LUNGIMEA MĂRIMII
       MØV    B,A     ;UNEI INTRĂRI ÎN TABELUL DE DATE
       PUSH   D       ;SALVEAZĂ ADRESA INTRĂRII Ki
       PUSH   H       ;SALVEAZĂ ADRESA INTRĂRII Ki+1
SØRT2: LDAX   D       ;ÎNCARCĂ ÎN REG. A UN OCTET AL IN-
                      ;TRĂRII Ki
       CMP    M       ;ȘI-L COMPARĂ CU OCTETUL CORESPUN-
                      ;ZĂTOR DIN Ki+1
       JC     SØRT1   ;SALT DACĂ ESTE MAI MIC, Ki < Ki+1
       JZ     SØRT1   ;SALT DACĂ ESTE EGAL Ki = Ki+1
       INX    H       ;INCREMENTEAZĂ CU 1 CONȚINUTUL
       INX    D       ;REGISTRELOR DUBLE HL ȘI DE
       DCR    B       ;DECREMENTEAZĂ CU 1 CONTOR LUNGIME
                      ;INTRARE
       JNZ    SØRT2   ;SALT CÎND CONTORUL NU ESTE ZERO
       PØP    H       ;REFACE ADRESA INTRĂRII Ki+1
       PØP    D       ;REFACE ADRESA INTRĂRII Ki
       LDA    LINTR   ;ÎNCARCĂ ÎN REGISTRUL B LUNGIMEA
       MØV    B,A     ;UNEI INTRĂRI ÎN TABELUL DE DATE
```

```
;SCHIMBĂ CONŢINUTUL INTRĂRII Ki CU CONŢINUTUL INTRĂRII Ki+1
SØRT5: MOV  C,M     ;SALVEAZĂ UN OCTET AL INTRĂRII Ki+1
       LDAX D       ;TRANSFERĂ ÎN LOCUL ACESTUIA OC-
       MØV  M,A     ;TETUL CORESPUNZĂTOR INTRĂRII Ki
       MØV  A,C     ;MEMOREAZĂ CONŢINUTUL REG. C LA
                    ;ADRESA DATĂ DE
       STAX D       ;CONŢINUTUL REG. D ŞI E, ADRESA IN-
                    ;TRĂRII Ki
       INX  H       ;INCREMENTEAZĂ CU 1 CONŢINUTUL
       INX  D       ;REGISTRELOR DUBLE HL ŞI DE
       DCR  B       ;DECREMENTEAZĂ CU 1 CONTOR LINTR
       JNZ  SØRT5   ;SALT CÎND CONTORUL NU ESTE ZERO
       MVI  A,1     ;POZIŢIONEAZĂ ÎN 1
       STA  CPRM    ;CONTORUL DE INVERSIUNI
       JMP  SØRT3
SØRT1: PØP  H       ;REFACE ADRESA INTRĂRII Ki+1
       PØP  D       ;REFACE ADRESA INTRĂRII Ki
       LDA  LINTR   ;CALCULEAZĂ ADRESA INTRĂRILOR UR-
                    ;MĂTOARE CE
       INX  D       ;VOR FI ANALIZATE, PRIN INCREMENTA-
                    ;REA CELOR
       INX  H       ;DOUĂ REGISTRE DUBLE DE ŞI HL CU
                    ;UN NUMĂR DE
       DCR  A       ;OCTEŢI EGAL CU CEL AFECTAT UNEI
                    ;INTRĂRI
       JNZ  $-3     ;RELUAREA OPERAŢIEI DE INCREMEN-
                    ;TARE
SØRT3: LDA  I       ;INCREMENTEAZĂ CU 1 NUMĂRUL DE IN-
                    ;TRĂRI
       INR  A       ;CHESTIONATE DIN TABELA DE DATE
       STA  I       ;ŞI-L MEMOREAZĂ
       PØP  B       ;REFACE CONŢINUTUL REGISTRULUI B,
                    ;NRI
       PUSH B       ;SALVEAZĂ CONŢINUTUL REG. B
       DCR  B       ;(B) ← NRI-1
       CMP  B       ;COMPARĂ I CU NRI-1
       JC   SØRT2-6 ;SALT DACĂ I ESTE MAI MIC DECÎT NRI-1
       LDA  CPRM    ;ÎNCARCĂ ÎN ACUMULATOR CONTOR IN-
                    ;VERSIUNI
       CPI  0       ;S-A TERMINAT SORTAREA?
       PØP  B       ;REFACE CONŢINUTUL REG. B, NRI, ŞI
                    ;REG. C
       PØP  H       ;REFACE CONŢINUTUL REG. H ŞI L, ADRE-
                    ;SA TABELULUI
       RZ           ;DA, REVENIRE ÎN PROGRAMUL PRIN-
                    ;CIPAL
       PUSH H       ;SALVEAZĂ CONŢINUTUL REG. H ŞI L
```

```
        PUSH B          ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
        JMP  SØRT4
I:      DS   1          ;CONTOR INTRĂRI PARCURSE
CPRM:   DS   1          ;INDICATOR INVERSIUNI
LVALI:  DS   1
LINTR:  DS   1
```

5.4.9. ARITMETICA ÎN COD BCD

5.4.9.1. Reprezentarea numerelor

În multe aplicații este necesar să se lucreze cu numere care au lungimea mai mare decît a unui cuvînt de 8 biți, admis de către microprocesor. Aceste *numere multi-octet* sînt privite ca un șir de octeți memorați în locații succesive de memorie, începînd cu octetul cel mai puțin semnificativ și terminînd cu octetul cel mai semnificativ (fig. 5.15).

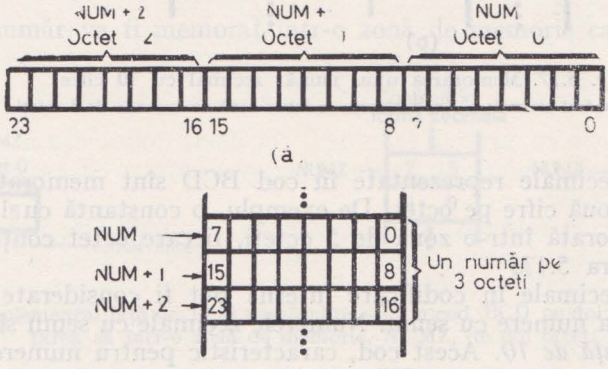


Fig. 5.15. Reprezentarea unui număr multioctet (a) în locații succesive de memorie (b).

Reprezentarea generală admisă pentru numere zecimale este codul BCD (Binary-Coded-Decimal). Aceasta constă în codificarea fiecărei cifre a numărului zecimal printr-un cod echivalent de 4 biți (fig. 5.16).

Cifră zecimală	Cod BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

(a)

Număr zecimal	Codificare BCD			
23	0010	0011		
99	1001	1001		
1892	0001	1000	1001	0010
354	0011	0101	0100	
9999	1001	1001	1001	1001

(b)

Fig. 5.16. Codificare BCD:
a — codul BCD al cifrelor zecimale; *b* — codificarea unor numere zecimale în BCD

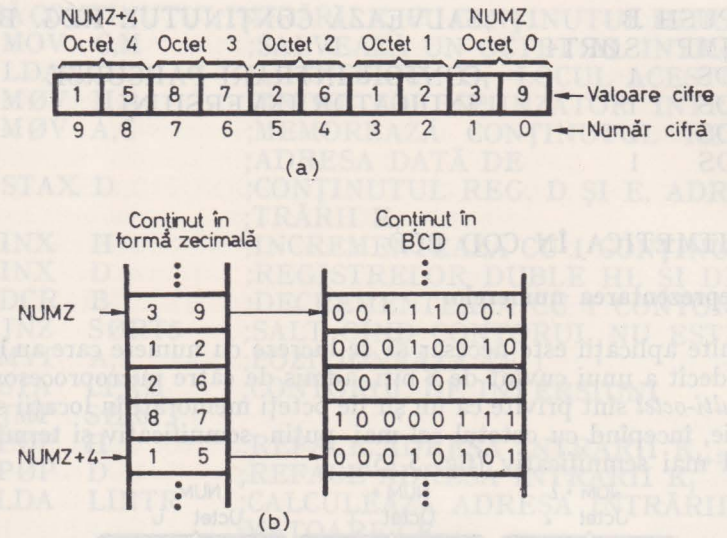


Fig. 5.17. Memorarea unui număr zecimal cu 10 cifre:
a — numărul zecimal; b — reprezentarea sa într-o zonă de memorie de 5 octeți.

Numerele zecimale reprezentate în cod BCD sînt memorate în locații succesive, cîte două cifre pe octet. De exemplu, o constantă cu 10 cifre zecimale va fi memorată într-o zonă de 5 octeți, fiecare octet conținînd cîte 2 cifre, ca în figura 5.17.

Numerele zecimale în codificare internă pot fi considerate ca numere fără semn, sau ca numere cu semn. Numerele zecimale cu semn sînt utilizate în *complement față de 10*. Acest cod, caracteristic pentru numerele cu semn în cod BCD, este asemănător codificării în complement față de 2 a numerelor reprezentate în cod binar cu semn (vezi § 5.3.3).

În situația în care un număr zecimal cu n cifre este considerat ca număr cu semn în complement față de 10 în cod BCD, primele n cifre reprezintă mărimea numărului, iar cifra a $n + 1$-a, semnul. Codificarea cifrei de semn, SZ, în cod BCD este:

SZ = 0000 pentru numere pozitive și zero
1001 pentru numere negative.

Dacă se lucrează cu numere zecimale de 3 cifre cu semn, primele 3 cifre reprezintă mărimea numărului, iar a patra, semnul (fig. 5.19).

În general, reprezentarea numerelor zecimale în cod BCD cu semn necesită o cifră de semn și un număr impar de cifre pentru număr. Dacă numărul zecimal cu semn are un număr par de cifre, pentru reprezentarea sa se va adăuga în poziția cea mai semnificativă a numărului codul cifrei zero și apoi codul cifrei de semn.

Exemplu. Numărul zecimal +7853 este reprezentat în cod BCD cu semn prin numărul 007853.

Reprezentarea numerelor zecimale cu semn în complement față de 10 se realizează după cum urmează:

— pentru numerele pozitive, cifrelor care reprezintă mărimea numărului li se adaugă în poziția cea mai semnificativă a acestuia cifra de semn, vezi exemplul de mai sus;

— pentru numerele negative, se calculează complementul față de 10; De exemplu, în cazul numerelor reprezentate pe 2 octeți sînt disponibile 4 poziții zecimale, dintre care 3 cifre sînt rezervate valorii și o cifră semnului. Complementul față de 10 se obține scăzînd în zecimal valoarea absolută a numărului din 10^4. Astfel, reprezentarea în complement față de 10 a numărului — 75 în cod BCD pe doi octeți este 9925, deoarece:

$$\begin{array}{rl} 10^4 = & 10000 \\ -75 = & -\ 00075 \\ \hline & 9925 \end{array}$$

Acest număr va fi memorat într-o zonă de memorie ca în figura 5.18.

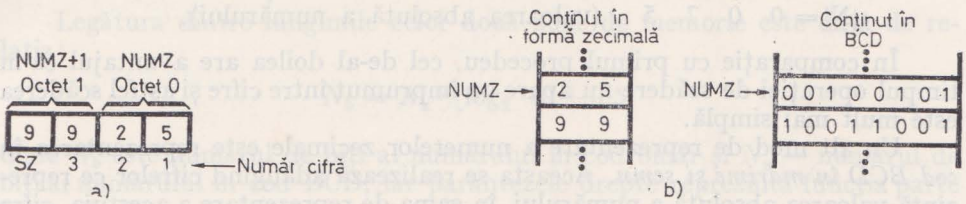


Fig. 5.18. Complementul față de 10 al numărului —75 în cod BCD pe doi octeți (a); reprezentarea sa într-o zonă de memorie, NUMZ, de doi octeți (b).

Gama numerelor zecimale cu semn în complement față de 10 ce pot fi codificate în cod BCD pe doi octeți este indicată în figura 5.19.

Număr zecimal	Complement față de 10
+999	0999
+998	0998
.	.
.	.
.	.
+1	0001
0	0000
−1	9999
−2	9998
.	.
.	.
.	.
−999	9001
−1000	9000

(a)

format zecimal	0	9	9	9
format BCD	0000	1001	1001	1001

(b)

format zecimal	9	0	0	0
format BCD	1001	0000	0000	0000

(c)

Fig. 5.19. Codificarea în cod BCD a numerelor zecimale cu semn:

a — complementul față de 10 al numerelor pe 2 octeți;
b — cel mai mare număr zecimal pozitiv cu semn pe 2 octeți;
c — cel mai mare, în valoare absolută, număr zecimal negativ pe 2 octeți, —1 000.

Valoarea absolută a unui număr negativ exprimat în complement față de 10 se poate obține prin scăderea zecimală a numărului din zero, neglijînd în final împrumutul, vezi exemplul 5-7, sau prin scăderea fiecărei cifre, inclusiv cifra de semn, din 9 și adunarea rezultatului cu 1, vezi exemplul 5-8.

Exemplul 5-7. Dacă numărul N exprimat în complement față de 10 în cod BCD este 9925, atunci valoarea absolută a acestuia se poate obține în felul următor:

$$\begin{array}{rr} & 0\ 0\ 0\ 0\ - \\ N = & 9\ 9\ 2\ 5 \\ \hline & 0\ 0\ 7\ 5 \end{array}$$ (valoarea absolută a numărului N)

Exemplul 5-8. Valoarea absolută a numărului 9925, ce reprezintă complementul față de 10 al unui număr zecimal, se obține prin scăderea fiecărei cifre a acestuia din cifra 9 și adunarea în cod BCD a rezultatului obținut cu 1:

$$\begin{array}{rr} & 9\ 9\ 9\ 9\ - \\ N = & 9\ 9\ 2\ 5 \\ \hline & 0\ 0\ 7\ 4\ + \\ & \qquad 1 \\ \hline |N| = & 0\ 0\ 7\ 5 \end{array}$$ (se incrementează cu 1 în cod BCD) (valoarea absolută a numărului).

În comparație cu primul procedeu, cel de-al doilea are avantajul că în timpul operației de scădere nu apare un împrumut între cifre și astfel scăderea este mult mai simplă.

Un alt mod de reprezentare a numerelor zecimale este reprezentarea în *cod BCD în mărime și semn.* Aceasta se realizează adăugînd cifrelor ce reprezintă valoarea absolută a numărului, în gama de reprezentare a acestuia, cifra de semn, ca în figura 5.20. Operațiile de înmulțire și împărțire în aritmetica zecimală prezentată folosesc numerele reprezentate în cod BCD în mărime și semn.

Număr zecimal	Reprezentarea acestuia în cod BCD în mărime și semn
+999	0999
+998	0998
.	.
.	.
.	.
+1	0001
0	0000
−1	9001
−2	9002
.	.
.	.
.	.
−998	9998
−999	9999

Fig. 5.20. Codificarea numerelor zecimale în cod BCD în mărime și semn pe doi octeți

Semnul numerelor, fie că este exprimat în cod complement față de 2, fie că este exprimat în cod BCD în complement față de 10 sau în cod BCD în mărime și semn, poate fi determinat prin testarea celui mai semnificativ bit al celui mai semnificativ octet al numărului.

5.4.9.2. Conversia unui număr din cod binar în cod BCD și invers

Reprezentarea numerelor în formă binară este frecvent folosită deoarece operațiile aritmetice sînt mai ușor de implementat și executat în cod binar. Pe de altă parte, în unele situații, numerele sînt mult mai ușor de exprimat în formă zecimală, fie că sînt date de intrare, fie că sînt rezultatele unei prelucrări și trebuie editate. Acest lucru implică necesitatea conversiei numerelor din cod BCD în cod binar și invers.

În prezentul paragraf se consideră că numerele ce trebuie convertite sînt numere fără semn. Aceste numere sînt memorate în două zone de memorie diferite, care se identifică cu ajutorul simbolurilor NUM pentru numerele în cod binar și NUMZ pentru numerele în cod BCD.

Legătura dintre lungimile celor două zone de memorie este dată de relația:

$$N_b = N_z \cdot [\log_2 10 + 0{,}5]$$

unde N_b este numărul de biți ai numărului în cod binar și N_z — numărul de biți ai numărului în cod BCD, iar parantezele drepte reprezintă funcția parte întreagă.

5.4.9.2.1. *Conversia unui număr din cod binar în cod BCD*

Întrucît incrementarea și decrementarea sînt operații ușor realizabile pentru numere oricît de mari, un prim algoritm de conversie al numerelor din cod binar în cod BCD se bazează pe aceste operații. Tehnica de conversie se mai numește și *conversie prin numărare*. Numărul în cod binar, NUM, este decrementat pînă cînd devine zero, iar numărul în cod BCD, NUMZ, este incrementat cu unu la fiecare operație de decrementare. Cu ajutorul instrucțiunii DAA aplicată numărului NUMZ, după fiecare operație de incrementare se va obține numărul convertit în cod BCD (vezi algoritmul din fig. 5.21*a*).

Pentru a putea programa algoritmul amintit vom prezenta subrutina INBCD, ce incrementează un număr în cod BCD conținut într-o zonă de memorie a cărei adresă este indicată de conținutul registrelor H și L, lungimea sa în octeți fiind egală cu conținutul registrului B. Cu ajutorul acestei subrutine pot fi incrementate numere pozitive care au maximum 255 × 2 cifre. În caz de depășire a lungimii numărului în cod BCD, programatorul va fi avertizat printr-un mesaj de eroare.

Subrutina INBCD alterează conținutul registrelor A, B și C, precum și al indicatorilor de condiții.

```
INBCD:  STC             ;POZIȚIONEAZĂ CY, CY = 1
        PUSH  H         ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
        MØV   A,M       ;TRANSFERĂ UN OCTET DIN MEMORIE
                        ;ÎN REG. A
        INR   A         ;ȘI-L INCREMENTEAZĂ CU 1
        DAA;            ;CORECTEAZĂ REZULTATUL OBȚINUT
        MØV   M,A       ;MEMOREAZĂ CONȚINUTUL ACUMULA-
                        ;TORULUI
        JNC   REV       ;REVENIRE DACĂ NU S-A PRODUS TRANS-
                        ;PORT ÎN CY
REL:    INX   H         ;INCREMENTEAZĂ ADRESA NUMĂRULUI
                        ;CU 1
        DCR   B         ;DECREMENTEAZĂ CONTOR LUNGIME
                        ;NUMĂR CU 1
        JZ    ER        ;SALT DACĂ Z = 1, S-A PRODUS TRANS-
                        ;PORT ÎN AFARA BITULUI CEL MAI SEM-
                        ;NIFICATIV AL NUMĂRULUI, NUMZ ESTE
                        ;PREA MIC
        MØV   A,M       ;INCREMENTEAZĂ CU 1 URMĂTORUL OC-
        ADI   1         ;TET AL NUMĂRULUI ÎN COD BCD
        JMP   REL-5
REV:    PØP   H         ;REFACE CONȚINUTUL REGISTRULUI
                        ;DUBLU HL
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
ER:     CALL  EDMES     ;EDITEAZĂ MESAJ DE EROARE
        PØP   H         ;REFACE CONȚINUTUL REGISTRULUI
                        ;DUBLU HL
        RET
EDMES:  LXI   H,ADMES   ;ÎNCARCĂ ÎN REG. H ȘI L ADRESĂ
                        ;MESAJ EROARE
        MVI   B,LMES    ;ÎNCARCĂ ÎN REG. B LUNGIME MESAJ
EDMS1:  JMP   TIPMS     ;EDITEAZĂ MESAJUL (vezi § 5.4.1)
ADMES:  DB              ;MESAJ EROARE
LMES    EQU   $-ADMES;LUNGIME MESAJ
```

Subrutina CBCD1 convertește un număr din cod binar într-un număr în cod BCD pe baza algoritmului indicat în figura 5.21*a*. Această subrutină necesită următoarele date de intrare: în registrele H și L adresa zonei de memorie unde se obține numărul convertit în cod BCD, NUMZ; în registrele D și E adresa zonei de memorie unde se află numărul în cod binar, NUM; în registrul B lungimea celor 2 zone de memorie. Se presupune că cele 2 zone de memorie NUM și NUMZ au aceeași lungime. La ieșirea din această subrutină va fi alterat conținutul registrului A și al indicatorilor de condiții.

```
CBCD1: PUSH  H      ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
       PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
       CALL  INMEM  ;INIȚIALIZEAZĂ ZONA DE MEMORIE AFEC-
                    ;TATĂ NUMĂRULUI ÎN COD
                    ;BCD-NUMZ (VEZI § 5.4.9.5)
       PØP   B      ;REFACE CONȚINUTUL REG. B ȘI C
       PØP   H      ;REFACE CONȚINUTUL REG. H ȘI L
       PUSH  D      ;SALVEAZĂ CONȚINUTUL REG. D ȘI E
       PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
REL1:  LDAX  D      ;VERIFICĂ DACĂ NUMĂRUL
       CPI   0      ;BINAR ESTE EGAL CU ZERO
       JNZ   $+12   ;SALT DACĂ NU ESTE EGAL
       INX   D      ;INCREMENTEAZĂ CU 1 CONȚINUTUL REG.
                    ;DUBLU DE
       DCR   B      ;DECREMENTEAZĂ CU 1 CONTOR LUNGIME
                    ;NUMĂR
       JNZ   REL1   ;SALT CÎND CONTORUL NU ESTE ZERO
       PØP   B      ;REFACE CONȚINUTUL REG. B ȘI C
       PØP   D      ;REFACE CONȚINUTUL REG. D ȘI E
       RET;         ;IEȘIRE PE NUMĂR BINAR EGAL CU 0
       PØP   D      ;REFACE CONȚINUTUL REG. D ȘI E
       PUSH  D      ;SALVEAZĂ CONȚINUTUL REG. D ȘI E
       PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
REL3:  LDAX  D      ;DECREMENTEAZĂ CU 1 NUMĂRUL
       SUI   1      ;BINAR NUM
       STAX  D      ;ȘI-L MEMOREAZĂ
       JNC   REL2   ;SALT DACĂ NU S-A PRODUS UN ÎMPRUMUT
       INX   D      ;INCREMENTEAZĂ CU 1 CONȚINUTUL REG.
       JMP   REL3   ;D ȘI E
REL2:  CALL  INBCD  ;INCREMENTEAZĂ ÎN BCD NUMĂRUL NUMZ
       PØP   B      ;REFACE CONȚINUTUL REG. B ȘI C
       PØP   D      ;REFACE CONȚINUTUL REG. D ȘI E
       JMP   REL1-2
```

Un algoritm mai rapid de conversie a unui număr din cod binar în cod BCD se bazează pe utilizarea instrucțiunii DAA pentru corecția zecimală a rezultatului sumei binare a 2 numere în cod BCD. Algoritmul consideră un număr în cod binar ca un polinom ce are drept coeficienți biții acestuia:

$$NUM = b_7 \cdot 2^7 + b_6 \cdot 2^6 + \ldots + b_1 \cdot 2 + b_0 \qquad (5.3)$$

Pentru simplitate se consideră că numărul în cod binar ocupă un octet. Relația (5.3) mai poate fi scrisă:

$$NUM = ((((((b_7 \cdot 2 + b_6) \cdot 2 + \ldots + b_1) \cdot 2 + b_0 \qquad (5.4)$$

Efectuînd operațiile indicate în relația (5.4) direct în cod BCD se va obține un algoritm iterativ de conversie în cod BCD a unui număr în cod binar.

$$NUMZ = ((((((b_7 \cdot 2 + b_6) \cdot 2 + \ldots + b_1) \cdot 2 + b_0 \qquad (5.5)$$

unde b_7, b_6, ..., b_0 sînt biții numărului în cod binar, NUM. În expresia (5.5) toate cantitățile sînt exprimate în cod BCD. Decurge un algoritm de conversie a unui număr în cod binar, NUM, cu NOBIN octeți, într-un număr în cod BCD, NUMZ, cu NOZEC octeți, prezentat în figura 5.21*b*. NBBIN indică numărul de biți ai numărului binar ce trebuie convertit. NOZEC trebuie ales suficient de mare pentru a putea exprima rezultatul conversiei.

Exemplu. Dacă NØBIN = 2, atunci NBBIN = 16. Conversia în cod BCD va necesita execuția a 16 bucle de program ($NUMZ \times 2 + b_i$). Fiecare valoare succesivă a lui b_i se obține prin deplasarea cu o poziție spre stînga a tuturor biților numărului în cod binar, NUM. Bitul cel mai semnificativ al numărului în cod binar, NUM, va fi transferat la fiecare deplasare în indicatorul de transport CY. În continuare, conținutul indicatorului CY, ce reprezintă bitul b_i, este adunat conform algoritmului cu NUMZ:

$$NUMZ \leftarrow NUMZ \times 2 + (CY),$$

folosind adunarea binară urmată de execuția instrucțiunii DAA.

Subrutina CBCD2 convertește un număr din cod binar în cod BCD pe baza algoritmului enunțat mai sus (fig. 5.21 *a*). Această subrutină permite

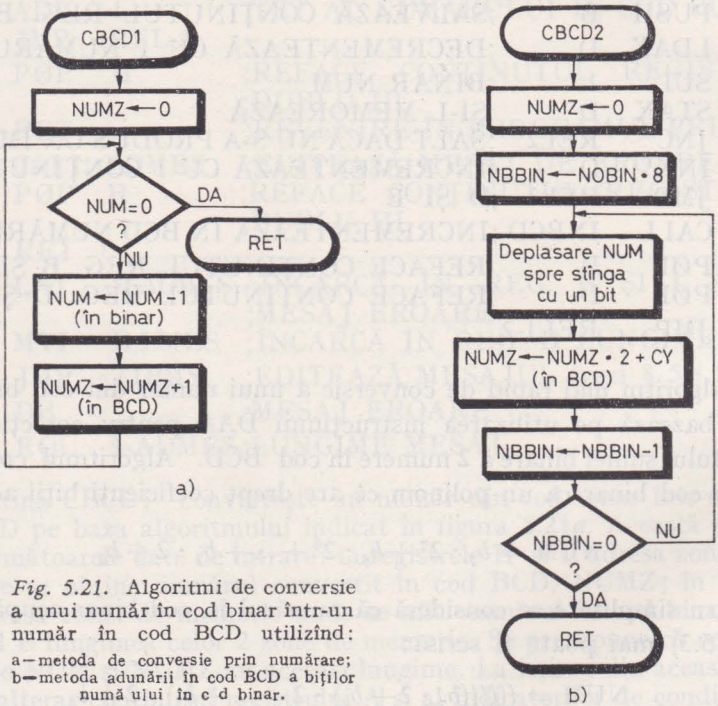


Fig. 5.21. Algoritmi de conversie a unui număr în cod binar într-un număr în cod BCD utilizînd:

a—metoda de conversie prin numărare;
b—metoda adunării în cod BCD a biților numă ului în c d binar.

conversia unui număr în cod binar cu maximum 32 octeți. Datele ce trebuie transmise de către programul apelant acestei subrutine sînt conținute în registrele: H și L — adresa zonei de memorie unde se obține numărul în cod BCD; D și E — adresa zonei de memorie unde se află numărul în cod binar; B — numărul de octeți ai numărului binar, NBIN; C — numărul de octeți afectați zonei de memorie pentru numărul în cod BCD, NOZEC. La ieșire va fi afectat conținutul registrului A și al indicatorilor de condiții.

```
NBBIN:  DS    1
CBCD2:  PUSH  H        ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
        PUSH  B        ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
        MØV   B,C      ;INIȚIALIZEAZĂ ZONA DE MEMORIE
        CALL  INMEM;   ;AFECTATĂ NUMĂRULUI ÎN COD
                       ;BCD-NUMZ (VEZI § 5.4.9.5).
        PØP   B        ;REFACE CONȚINUTUL REG. B ȘI C
        MØV   A, B     ;(A) ← NØBIN
        RLC            ;DUPĂ DEPLASAREA CONȚINUTULUI
        RLC            ;ACUMULATORULUI
        RLC            ;SPRE STÎNGA CU 3 BIȚI SE OBȚINE
                       ;ÎN ACUMULATOR NUMĂRUL DE BIȚI
        STA   NBBIN    ;AI NR. BINAR CE TREBUIE CONVER-
                       ;TIT.
        PØP   H        :REFACE CONȚINUTUL REG. H ȘI L
        STC            ;POZIȚIONEAZĂ CY (CY=1)
        PUSH  D        ;SALVEAZĂ CONȚINUTUL REG. DUBLE
        PUSH  B        ;DE ȘI BC
REL1:   CALL  DSBIN    ;DEPLASEAZĂ NUMĂRUL SPRE STÎNGA
                       ;CU UN BIT
        PØP   B        ;REFACE CONȚINUTUL REG. DUBLU
                       ;BC
        PUSH  H        ;REFACE CONȚINUTUL REG. DUBLU HL
        PUSH  B        ;SALVEAZĂ CONȚINUTUL REG. DUBLU
                       ;BC
REL2:   MVI   A,0      ;ȘTERGE ACUMULATORUL
        RAL            ;TRANSFERĂ (CY) ÎN BITUL 0 AL
                       ;ACUMULATORULUI
        ADD   M        ;ADUNĂ (A) CU CONȚINUTUL UNEI
        DAA            ;LOCAȚII DE MEMORIE ÎN COD BCD
        MØV   M,A      ;MEMOREAZĂ REZULTATUL OBȚINUT
REL5:   JNC   REL4     ;DACĂ NU S-A PRODUS UN TRANSPORT
;ÎN AFARA BITULUI CEL MAI SEMNIFICATIV AL ACUMULATO-
;RULUI SE PRODUCE SALTUL
        INX   H        ;INCREMENTEAZĂ CONȚINUTUL REG.
                       ;DUBLU HL CU 1
        DCR   C        ;S-A PRODUS UN TRANSPORT ÎN
                       ;AFARA BITULUI CEL; MAI SEMNIFI-
                       ;CATIV AL NUMĂRULUI NUMZ?
```

```
        JZ      ER          ;DA, EROARE
        MØV     A,M         ;NU, ADUNĂ CU 1 DIN CY CONȚINUTUL
        ACI     0           ;URMĂTORULUI OCTET AL NUMĂRU-
                            ;LUI NUMZ
        JMP     REL5-2
REL4:   PØP     B           ;REFACE CONȚINUTUL REGISTRELOR
        PØP     H           ;DUBLE BC, HL ȘI DE
        PØP     D
        LDA     NBBIN       ;ÎNCARCĂ ÎN A NUMĂRUL DE BIȚI AI
        DCR     A           ;NUMĂRULUI BINAR CE AU
        STA     NBBIN       ;RĂMAS DE PRELUCRAT
        JNZ     REL1-2      ;SALT DACĂ NBBIN=0
        RET                 ;REVENIRE ÎN PROGRAMUL PRINCIPAL
ER:     .....               ;EDITEAZĂ MESAJUL DE EROARE, DE
        .....               ;DEPĂȘIRE A ZONEI DE MEMORIE
                            ;NUMZ
        PØP     B           ;REFACE CONȚINUTUL REGISTRELOR
        PØP     H           ;DUBLE, BC, HL ȘI DE
        PØP     D
        RET                 ;REVENIRE ÎN PROGRAMUL PRIN-
                            ;CIPAL
```

Subrutina DSBIN execută deplasarea spre stînga cu un bit a unui număr multioctet în cod binar, memorat într-o zonă a cărei adresă este indicată de conținutul registrelor D și E și de lungime în octeți dată de conținutul registrului B. La ieșire, subrutina alterează conținutul registrelor A, B și al indicatorilor de condiții.

```
DSBIN:  PUSH D              ;SALVEAZĂ ADRESA NUMĂRULUI
        XRA  A              ;ȘTERGE CY, CY=0
        LDAX D              ;(A) ← ((D)(E))
        RAL                 ;DEPLASEAZĂ (A) SPRE STÎNGA PRIN
                            ;CY
        STAX D              ;MEMOREAZĂ CONȚINUTUL ACUMULA-
                            ;TORULUI
        INX  D              ;(D)(E) ← (D)(E)+1
        DCR  B              ;DECREMENTEAZĂ CU 1 CONTOR
                            ;LUNGIME NUMĂR
        JNZ  DSBIN+2        ;SALT DACĂ NU S-A TERMINAT
                            ;OPERAȚIA DE DEPLASARE.
        PØP  D              ;REFACE CONȚINUTUL REG. D ȘI E
        RET                 ;REVENIRE ÎN PROGRAMUL PRINCIPAL
```

Conversia numerelor binare cu semn reprezentate în complement față de 2 în cod BCD în mărime și semn se va face în felul următor: valoarea absolută a numărului este convertită în cod BCD cu ajutorul unuia din cei 2 algoritmi descriși mai sus; numărului rezultat, NUMZ, i se va anexa cifra de semn, corespunzătoare numărului binar cu semn convertit.

5.4.9.2.2. *Conversia unui număr din cod BCD în cod binar*

Conversia unui număr din cod BCD în cod binar se poate realiza folosind un algoritm asemănător cu cel descris în figura 5.21*a*, prin decrementarea numărului în cod BCD pînă la zero și incrementarea corespunzătoare a numărului în cod binar de la zero pînă la valoarea convertită. Operația de decrementare a unui număr în cod BCD este mai dificil de realizat decît decrementarea unui număr în cod binar. Acest lucru se poate evita prin scăderea binară a numărului în cod BCD dintr-un număr cu atîtea cifre de 9 cîte poziții are numărul în cod BCD, rezultatul fiind apoi incrementat cu ajutorul subrutinei INBCD pînă cînd toate cifrele devin 9 (fig. 5.22*a*).

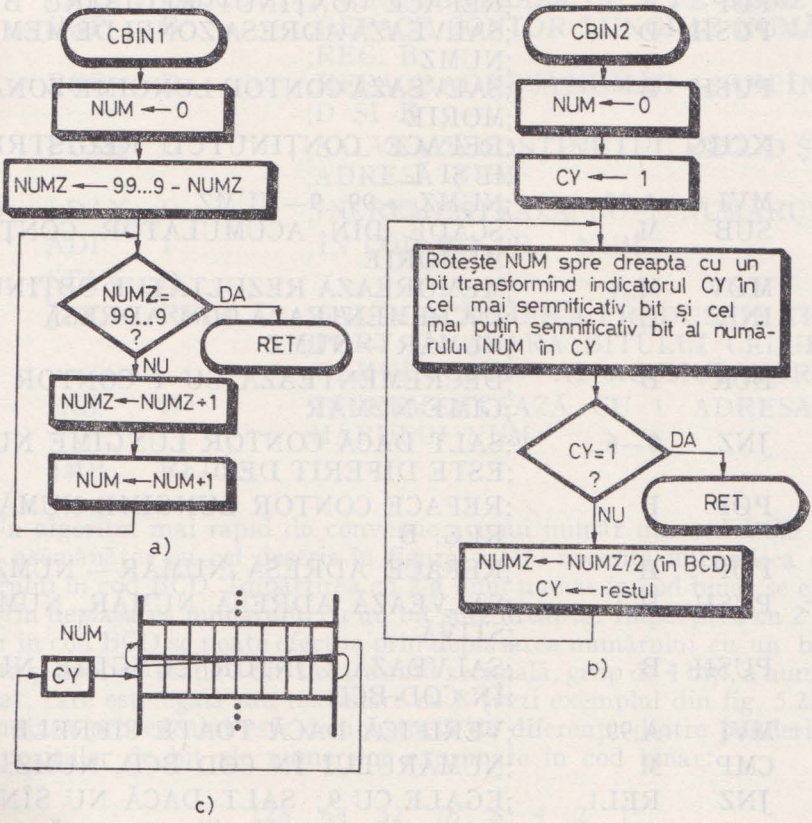


Fig. 5.22. Algoritmi de conversie a unui număr în cod BCD într-un număr în cod binar utilizînd:

a—metoda de conversie prin numărare; b—metoda împărțirii cu 2; c—definirea operației de rotire a unui număr în cod binar, NUM, cu un bit spre stînga.

Subrutina CBIN1 convertește un număr din cod BCD în cod binar pe baza algoritmului descris în organigrama din figura 5.22*a*. Datele de intrare necesare acestei subrutine sînt conținute în registrele: H și L — adresa zonei de memorie unde se află numărul în cod BCD, NUMZ; D și E — adresa zonei de memorie unde se obține numărul binar, NUM; B — lungimea celor două zone de memorie în octeți. Cu ajutorul acestei subrutine se pot converti în cod binar numere exprimate în cod BCD cu maximum 255 ×2 cifre. Subrutina alterează conținutul acumulatorului și al indicatorilor de condiții.

```
CBIN1: PUSH  D        ;SALVEAZĂ ADRESA ZONEI DE MEMORIE
                      ;NUM
       PUSH  B        ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
       XCHG           ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
       CALL  INMEM    ;INIȚIALIZEAZĂ ZONA DE MEMORIE
;AFECTATĂ NUMĂRULUI ÎN COD BINAR — NUM
       PØP   B        ;REFACE CONȚINUT REGISTRU B ȘI C
       PUSH  D        ;SALVEAZĂ ADRESA ZONEI DE MEMORIE
                      ;NUMZ
       PUSH  B        ;SALVEAZĂ CONTOR LUNGIME ZONĂ ME-
                      ;MORIE
       XCHG           ;REFACE CONȚINUTUL REGISTRELOR
                      ;H ȘI L
       MVI   A,99     ;NUMZ → 99...9—NUMZ
       SUB   M        ;SCADE DIN ACUMULATOR CONȚINUT
                      ;MEMORIE
       MØV   M,A      ;MEMOREAZĂ REZULTATUL OBȚINUT
       INX   H        ;INCREMENTEAZĂ CU 1 ADRESĂ
                      ;NUMĂR — NUMZ
       DCR   B        ;DECREMENTEAZĂ CU 1 CONTOR LUN-
                      ;GIME NUMĂR
       JNZ   $—6      ;SALT DACĂ CONTOR LUNGIME NUMĂR
                      ;ESTE DIFERIT DE 0
       PØP   B        ;REFACE CONTOR LUNGIME NUMĂR ÎN
                      ;REG. B
       PØP   H        ;REFACE ADRESĂ NUMĂR — NUMZ
REL4:  PUSH  H        ;SALVEAZĂ ADRESĂ NUMĂR, NUMZ ÎN
                      ;STIVĂ
       PUSH  B        ;SALVEAZĂ CONTOR LUNGIME NUMĂR
                      ;ÎN COD BCD
       MVI   A,99     ;VERIFICĂ DACĂ TOATE CIFRELE
       CMP   M        ;NUMĂRULUI ÎN COD BCD, NUMZ, SÎNT
       JNZ   REL1     ;EGALE CU 9; SALT DACĂ NU SÎNT
       INX   H        ;INCREMENTEAZĂ CU 1 ADRESĂ NUMĂR
                      ;— NUMZ
       DCR   B        ;DECREMENTEAZĂ CU 1 CONTOR LUNGI-
                      ;ME NUMĂR
```

```
        JNZ   $—6       ;RELUARE TEST CÎND CONTORUL ESTE
                        ;DIFERIT DE 0
        PØP   B         ;REFACE CONTOR LUNGIME NUMĂR ÎN
                        ;REG. B
        PØP   H         ;REFACE ADRESĂ NUMĂR — NUMZ ÎN
                        ;REG. H ŞI L
        PØP   D         :REFACE ADRESĂ NUMĂR — NUM ÎN
                        ;REG. D ŞI E
        RET             ;REVINE ÎN PROGRAMUL PRINCIPAL
REL1:   PØP   B         ;REFACE CONŢINUTUL REG. B — LUN-
                        ;GIME NUMĂR
        PØP   H         ;REFACE CONŢINUTUL REG. H ŞI L —
                        ;ADRESĂ NUMZ
        PUSH  Б         ;SALVEAZĂ CONŢINUTUL REG. B ŞI C
        CALL  INBCD     ;INCREMENTEAZĂ CU 1 PE NUMZ
        PØP   B         ;REFACE CONTOR LUNGIME NUMĂR ÎN
                        ;REG. B
        PØP   D         ;REFACE ADRESĂ NUMĂR —NUM ÎN REG
                        ;D ŞI E
        PUSH  D         ;SALVEAZĂ CONŢINUTUL REG. D ŞI E —
                        ;ADRESĂ NUM
REL3:   LDAX  D         ;INCREMENTEAZĂ CU 1 NUMĂRUL
        ADI   1         ;ÎN COD BINAR — NUM
        STAX  D
        JNC   REL4      ;SALT DACĂ NU S-A PRODUS UN TRANS-
                        ;PORT ÎN AFARA BITULUI CELUI MAI
                        ;SEMNIFICATIV AL ACUMULATORULUI
        INX   D         ;INCREMENTEAZĂ CU 1 ADRESA NU-
                        ;MĂRULUI NUM
        JMP   REL3
```

Un algoritm mai rapid de conversie a unui număr din cod BCD în cod binar, asemănător cu cel descris în figura 5.21*b*, necesită împărţirea cu 2 a numărului în cod BCD. Împărţirea cu 2 a unui număr în cod binar se execută uşor prin deplasarea numărului cu un bit spre dreapta. Împărţirea cu 2 a unui număr în cod BCD se poate efectua prin deplasarea numărului cu un bit spre dreapta şi apoi se scade 3 din fiecare cifră zecimală, grup de 4 biţi, a numărului rezultat, care este egală sau mai mare cu 8 (vezi exemplul din fig. 5.23). Corectitudinea procedeului este demonstrată de diferenţa dintre ponderile asociate poziţiilor de bit ale numerelor exprimate în cod binar:

... 128 64 32 *16* 8 4 2 1

şi ponderile asociate poziţiilor de bit ale numerelor scrise în cod BCD:

... 80 40 20 *10* 8 4 2 1

Deplasarea unui 1 la dreapta din poziția subliniată va produce o împărțire cu 2 a unui număr în cod binar, deoarece bitul rezultat va avea ponderea egală cu 8. Pentru numărul în cod BCD ponderea corectă este 5 (10/2) și nu 8. Scăzînd 3 cînd există 1 în această poziție de bit, se face corecția dorită (vezi exemplul din fig. 5.23).

Număr zecimal	Număr în cod BCD				
5672	0101	0110	0111	0010	} deplasează numărul spre dreapta cu un bit
	0010	1011	0011	1001	
		−0011		−0011	} identifică cifrele egale sau mai mari cu cifra 8 și scade 3 unde este cazu
2836	0010	1000	0011	0110	

Fig. 5.23. Exemplu de împărțire cu 2 a unui număr în cod BCD

Pentru a înțelege raționamentul acestui algoritm de conversie, se consideră două numere egale în valoare, NUMZ și NUM, exprimate în cod BCD și, respectiv, în cod binar.

Dacă cele două numere sînt pare, atît NUM cît și NUMZ au bitul cel mai puțin semnificativ egal cu 0, iar dacă sînt impare au bitul cel mai puțin semnificativ egal cu 1. În consecință bitul cel mai puțin semnificativ al unui număr în cod binar este egal cu bitul similar al reprezentării în cod BCD a aceluiași număr și poate fi obținut direct. Prin împărțirea cu 2 a celor 2 numere și neglijarea resturilor se vor obține două numere egale în valoare, dar exprimate în coduri diferite. În baza raționamentului de mai sus, reiese că valoarea bitului celui mai puțin semnificativ al fiecăruia dintre cele 2 numere rezultate este aceeași. Dar bitul cel mai puțin semnificativ al numărului în cod binar rezultat este cel de-al doilea bit al numărului în cod binar inițial — NUM. Asemănător se pot obține toți biții numărului în cod binar, NUM, prin operații succesive de împărțire cu 2 a numărului în cod BCD, NUMZ, reținînd de fiecare dată restul împărțirii, care este bitul cel mai puțin semnificativ al numărului în cod BCD, în indicatorul de transport CY. După fiecare împărțire, conținutul indicatorului de transport, CY, va fi introdus, printr-o operație de rotire a numărului în cod binar, NUM, cu un bit spre dreapta, în bitul cel mai semnificativ al numărului în cod binar. Această operație este iterativă și se execută de NOBCD ×8 ori, unde NOBCD reprezintă numărul de octeți ai numărului în cod BCD ce trebuie convertit. În final, bitul cel mai puțin semnificativ al numărului în cod BCD, NUMZ, va deveni bitul cel mai puțin semnificativ al numărului în cod binar, NUM.

Un procedeu mai simplu de contorizare a numărului de iterații ce sînt necesare conversiei unui număr din cod BCD într-un număr în cod binar este următorul:

— se șterge zona de memorie afectată numărului în cod binar, NUM;

— cu ajutorul operației de rotire spre dreapta prin CY cu un bit a numărului în cod binar NUM, vezi subrutina DDBIN, se introduce în bitul cel

mai semnificativ al numărului conținutul indicatorului de transport, CY, care este inițial egalat cu 1;

— în cursul operației de conversie, acest bit va fi deplasat după fiecare împărțire cu 2 a numărului în cod BCD, fiind urmat de biții succesivi ai rezultatului;

— operația de conversie, în conformitate cu algoritmul descris mai sus, continuă pînă cînd bitul cel mai puțin semnificativ al rezultatului ajunge în poziția cea mai puțin semnificativă a numărului în cod binar, NUM. Sfîrșitul operației de conversie este marcat de primul transport egal cu 1 în CY din poziția cea mai puțin semnificativă a numărului în cod binar NUM.

Algoritmul este rezumat în organigrama din figura 5.22*b*, iar programul corespunzător acesteia, CBIN2, este prezentat în continuare. Cu ajutorul subrutinei CBIN2 se pot converti din cod binar în cod BCD numere cu maximum 255 ×2 cifre. Datele de intrare necesare acestei subrutine sînt indicate de conținutul registrelor: D și E — adresa zonei unde este memorat numărul în cod BCD ce trebuie convertit, NUMZ; H și L — adresa zonei de memorie unde se obține numărul în cod binar, NUM; C — lungimea celor două zone de memorie. Subrutina alterează conținutul registrelor A, B și al indicatorilor de condiții.

```
CBIN2:  PUSH  B         ;SALVEAZĂ CONȚINUTUL REGISTRELOR
                        ;B ȘI C
        XRA   A         ;ȘTERGE CONȚINUTUL ACUMULATORU-
                        ;LUI ȘI CY
        MØV   M,A       ;INTRODUCE ZERO ÎN ZONA DE MEMORIE
        INX   H         ;AFECTATĂ NUMĂRULUI ÎN COD BINAR
                        ;NUM
        DCR   C         ;DECREMENTEAZĂ CU 1 CONTOR LUN-
                        ;GIME NUMĂR
        JNZ   $—3       ;SALT DACĂ NU ESTE ZERO
        CMC             ;INTRODUCE 1 ÎN INDICATORUL DE
                        ;TRANSPORT
        PØP   B         ;REFACE CONȚINUTUL REGISTRELOR
                        ;B ȘI C
        PUSH  B         ;SALVEAZĂ LUNGIME NUMĂR ÎN STIVĂ
                        ;— REG. C
        CALL  DDBIN+5   ;INTRODUCE 1 ÎN BITUL CEL MAI SEM-
;NIFICATIV AL NUMĂRULUI ÎN COD BINAR — NUM
RELY:   PØP   B         ;REFACE CONTOR LUNGIME NUMĂR ÎN
                        ;REG. C
        CALL  DDBCD     ;ÎMPARTE NUMĂRUL ÎN COD BCD CU 2
        PUSH  B         ;SALVEAZĂ LUNGIME NUMĂR — REG. C
        CALL  DDBIN     ;DEPLASEAZĂ NUMĂRUL ÎN COD BINAR
;NUM CU UN BIT SPRE DREAPTA PRIN INTRODUCEREA
;INDICATORULUI DE TRANSPORT
```

```
;CY ÎN BITUL CEL MAI SEMNIFICATIV AL NUMĂRULUI ŞI A BITULUI
;CELUI MAI PUŢIN SEMNIFICATIV AL NUMĂRULUI ÎN INDICATO-
;RUL CY
        JNC     RELY    ;SALT DACĂ CY=0
        PØP     B       ;REFACE CONŢINUTUL REGISTRULUI C
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
```

Subrutina DDBIN deplasează cu un bit spre dreapta prin CY (fig. 5.22*c*) un număr în cod binar, aflat într-o zonă de memorie a cărei adresă este indicată de conţinutul registrelor H şi L, lungimea fiind egală cu conţinutul registrului C. La ieşire, bitul cel mai puţin semnificativ al numărului deplasat se află în CY, conţinutul registrelor A, B şi C şi al indicatorilor de condiţii fiind alterat.

```
DDBIN:  PUSH    PSW     ;SALVEAZĂ INDICATORUL DE TRANS-
                        ;PORT, CY
        MVI     B,0     ;ÎNCARCĂ IMEDIAT ÎN REG. B ZERO
        DAD     B       ;(H)(L) ← (H)(L)+(B)(C)
        PØP     PSW     ;REFACE CONŢINUTUL INDICATORULUI
                        ;DE TRANSPORT
        DCX     H       ;DECREMENTEAZĂ CU 1 ADRESA NU-
                        ;MĂRULUI
        MØV     A,M     ;ÎNCARCĂ ÎN ACUMULATOR UN OCTET
                        ;AL NUMĂRULUI
        RAR             ;A7 ← CY, A6 ← A7, ..., CY ← A0
        MØV     M,A     ;MEMOREAZĂ CONŢINUTUL ACUMULA-
                        ;TORULUI
        DCR     C       ;DECREMENTEAZĂ CU 1 CONTOR LUNGI-
                        ;ME NUMĂR
        JNZ     DDBIN+5 ;SALT DACĂ CONTORUL ESTE ZERO
        RET             ;REVENIRE ÎN PROGRAMUL CHEMĂTOR
```

Subrutina DDBCD deplasează un număr în cod BCD cu un bit spre dreapta prin indicatorul de transport, CY. Această operaţie este echivalentă cu împărţirea cu 2 a unui număr în cod BCD, unde restul împărţirii se reţine în indicatorul de transport, CY, şi cîtul în locul deîmpărţitului. Datele de intrare necesare acestei subrutine sînt conţinute în registrele: D şi E — adresa zonei de memorie unde se află numărul în cod BCD, NUMZ, ce trebuie împărţit cu 2; C — lungimea numărului în octeţi. La ieşire, indicatorul CY va conţine bitul cel mai puţin semnificativ al numărului iniţial adică restul împărţirii acestuia cu 2. Subrutina alterează conţinutul registrelor A, B şi al indicatorilor de condiţii.

```
DDBCD:  XRA     A       ;ŞTERGE INDICATORUL DE TRANSPORT
                        ;CY
        MØV     B,C     ;SALVEAZĂ CONŢINUTUL REGISTRULUI
                        ;C
        PUSH    B       ;SALVEAZĂ CONŢINUTUL REGISTRELOR
                        ;B ŞI C
```

```
        XCHG            ;TRANSFERĂ CONȚINUTUL REG. D ȘI E
                        ;ÎN REG. H ȘI L
        CALL    DDBIN   ;DEPLASEAZĂ NUMĂRUL ÎN COD BCD
                        ;CA UN NUMĂR ÎN COD
;BINAR CU UN BIT SPRE DREAPTA, PRIN INDICATORUL DE
;TRANSPORT, CY
        PUSH    PSW     ;SALVEAZĂ RESTUL DIN CY
        PØP     B       ;REFACE CONȚINUTUL REG. B ȘI C
        PUSH    B       ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
        PUSH    H       ;SALVEAZĂ ADRESA NUMĂRULUI —
                        ;NUMZ
RELX:   MØV     A,M     ;ÎNCARCĂ ÎN ACUMULATOR 2 CIFRE ALE
                        ;NUMĂRULUI
        MØV     C,A     ;SALVEAZĂ CONȚINUTUL ACUMULATO-
                        ;RULUI
        ANI     0FH     ;REȚINE CIFRA DE PONDERE INFERI-
                        ;OARĂ
        CPI     8       ;ESTE MAI MICĂ DECÎT 8?
        JC      $+5     ;DA, SALT
        SUI     3       ;NU, SCADE 3
        PUSH    PSW     ;SALVEAZĂ CIFRA
        MØV     A,C     ;TRANSFERĂ CONȚINUTUL REG. C ÎN
                        ;ACUMULATOR
        ANI     0F0H    ;REȚINE CIFRA DE PONDERE SUPERI-
                        ;OARĂ
        CPI     80H     ;CIFRA ESTE MAI MICĂ DECÎT 8?
        JC      $+5     ;DA, SALT
        SUI     3       ;NU, SCADE 3
        MØV     C,A     ;SALVEAZĂ CONȚINUTUL ACUMULATO-
                        ;RULUI
        PØP     PSW     ;ÎNCARCĂ ÎN ACUMULATOR PRIMA CI-
                        ;FRĂ
        ØRA     C       ;ÎNCARCĂ ÎN ACUMULATOR CELE DOUĂ
                        ;CIFRE
        MØV     M,A     ;MEMOREAZĂ CELE DOUĂ CIFRE
        INX     H       ;INCREMENTEAZĂ ADRESA NUMĂRULUI
                        ;CU 1
        DCR     B       ;AU FOST VERIFICATE TOATE CIFRELE
                        ;NUMĂRULUI ?
        JNZ     RELX    ;NU, SALT
        PØP     H       ;REFACE ADRESA NUMĂRULUI ÎN REG.
                        ;H ȘI L
        PØP     B       ;REFACE CONȚINUTUL REG. C
        PØP     PSW     ;REFACE RESTUL DIN CY
        XCHG            ;REFACE CONȚINUTUL INIȚIAL AL REG.
                        ;D, E, H ȘI L
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
```

5.4.9.3. Adunarea a 2 numere

Suma a două numere în cod BCD se execută utilizînd instrucțiunile de adunare binară a 2 octeți și instrucțiunea de corecție a rezultatului adunării a 2 numere în cod BCD — DAA. Poziționarea indicatorului de transport în funcție de rezultatul obținut permite adunarea a două numere în cod BCD mai lungi de două cifre.

Exemplu. Adunarea a 2 numere în cod BCD cu 4 cifre, 2985+4936= =7921, se execută în felul următor:

1. Se șterge indicatorul de transport, CY, și se adună binar primele 2 cifre, de pondere inferioară, ale celor 2 numere:

```
85 =  1000   0101
36 =  0011   0110
CY=             0
(A)= |-1011  |-1011
     |→CY=0  |→AC=0
```

2. În continuare, cu ajutorul instrucțiunii DAA, se corectează rezultatul operației anterioare. Deoarece cei mai puțin semnificativi 4 biți ai acumulatorului conțin o valoare mai mare decît 9, aceasta adună 6 la conținutul acumulatorului:

```
(A)= 1011   1011
            0110
(A)= 1100   0001
```

Acum valoarea celor mai semnificativi 4 biți ai acumulatorului este mai mare decît 9. În consecință valoarea acestor biți este adunată cu 6 de către DAA:

```
(A)=  1100   0001
      0110
(A)= |-0010   0001 = 21
     |→CY=1
```

Conținutul acumulatorului se memorează în zona afectată rezultatului, de regulă în locul primului operand.

3. Se adună următorul grup de 2 cifre al celor 2 numere în cod BCD și transportul din suma precedentă;

```
29 =  0010   1001
49 =  0100   1001
CY=             1
(A)= |0111  |0011 = 73
     |→CY=0 |→AC=1
```

4. Rezultatul adunării anterioare este corectat cu ajutorul instrucțiunii DAA. Deoarece AC = 1, conținutul acumulatorului se adună cu 6;

```
(A) =  0111    0011
               0110
       ---------------
(A) = |-0111    1001 = 79
      |→CY=0
```

Subrutina ADBCD execută adunarea a două numere în cod BCD pe baza algoritmului descris în exemplul anterior. Cele 2 numere pot avea maximum 255×2 cifre. Datele de intrare necesare acestei subrutine sînt indicate de conținutul registrelor: D și E — adresa zonei unde este memorat primul operand; H și L — adresa zonei unde este memorat cel de-al doilea operand; B — lungimea celor 2 numere de octeți. La ieșire, rezultatul este obținut în zona de memorie afectată primului operand. Subrutina alterează conținutul registrelor A, B și al indicatorilor de condiții.

```
ADBCD: XRA  A        ;ȘTERGE INDICATORUL DE TRANSPORT
                     ;CY=0
       PUSH D        ;SALVEAZĂ CONȚINUTUL REG. D ȘI E
       PUSH H        ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
       LDAX D        ;ADUNĂ CONȚINUTUL A DOI OCTEȚI DE
                     ;ACEEAȘI
       ADC  M        ;PONDERE ȘI TRANSPORTUL ANTERIOR
                     ;ÎN COD
       DAA           ;BCD ȘI-L MEMOREAZĂ ÎN ZONA DE
                     ;MEMORIE
       STAX D        ;AFECTATĂ PRIMULUI OPERAND
       INX  D        ;INCREMENTEAZĂ CU 1 ADRESA PRI-
                     ;MULUI OPERAND
       INX  H        ;INCREMENTEAZĂ CU 1 ADRESA CELUI
                     ;DE-AL DOILEA OP.
       DCR  B        ;DECREMENTEAZĂ CU 1 CONTOR
                     ;LUNGIME OPERANZI
       JNZ  ADBCD+3  ;SALT CÎND CONTORUL ESTE DIFERIT
                     ;DE ZERO
       PØP  H        ;REFACE CONȚINUTUL REG. H ȘI L
       PØP  D        ;REFACE CONȚINUTUL REG. D ȘI E
       RET           ;REVENIRE ÎN PROGRAMUL PRINCIPAL
```

La ieșirea din această subrutină conținutul indicatorului de transport, CY, va indica dacă în timpul operației de adunare s-a produs un transport în afara bitului cel mai semnificativ al rezultatului, care este interpretat ca transport zecimal.

5.4.9.4. Scăderea a 2 numere

Diferența a două numere zecimale în cod BCD se execută cu ajutorul operației de adunare în cod BCD, efectuată între descăzut și complementul față de 10 al scăzătorului.

De exemplu, diferența a 2 numere de două cifre în cod BCD, 56 și 34, se execută adunînd 56, în cod BCD, cu complementul față de 10 al numărului 34, 100—34=66, obținîndu-se 122, care trunchiat la 8 biți va da rezultatul corect, 22. Transportul generat, CY=1, arată că nu s-a produs un împrumut.

În general, scăderea a 2 numere de lungime variabilă în cod BCD se execută după următorul algoritm iterativ:

1. Indicatorul de transport este poziționat, CY=1.
2. Registrul A se încarcă cu 99H.
3. Conținutul registrului A se adună cu indicatorul de transport CY.
4. Un octet al descăzutului, la prima iterație octetul cel mai puțin semnificativ este scăzut în binar din conținutul acumulatorului, calculîndu-se astfel o porțiune de două cifre din complementul față de 10 al scăzătorului.
5. Conținutul acumulatorului este adunat în binar cu cele două cifre ale descăzutului de pondere egală cu cea a cifrelor scăzătorului folosite la punctul 4.
6. Rezultatul adunării precedente este corectat cu ajutorul instrucțiunii DAA. Dacă în urma execuției operațiilor de la punctele 5 și 6 indicatorul CY nu se poziționează, rezultă că scăderea celor două cifre s-a executat cu un împrumut la cifra de ordin superior acestora. Conținutul acumulatorului se va memora în zona afectată descăzutului;
7. Dacă operația de scădere nu s-a terminat, prelucrarea se reia de la punctul 2 pentru următoarele 2 cifre ale celor doi operanzi.

Exemplu. Scăderea a două numere zecimale în cod BCD de 4 cifre, 4358—1362=2996, bazată pe algoritmul descris mai sus, se execută după cum urmează:

1. CY ← 1.
2. (A) ← 99H.
3. (A) ← (A) +CY=99H+1=9AH.
4. Se scad cifrele de pondere inferioară ale scăzătorului din conținutul acumulatorului, calculîndu-se în acest fel complementul față de 10 al acestora:

$$\begin{array}{rl} (A) = & 1001\ \ 1010 + \\ \overline{62}H = & 1001\ \ 1110 \\ \hline & 1]\ 0011\ \ 1000 \end{array}$$

complementul față de 2 al numărului 62H

5. Conţinutul acumulatorului este adunat cu octetul de pondere inferioară al descăzutului:

$$\begin{array}{rll} (A) = & 0011 & 1000+ \\ 58H = & 0101 & 1000 \\ \hline & 1001 & 0000 \\ & \longrightarrow CY=0 & \longrightarrow AC=1 \end{array}$$

CY=0 arată că s-a făcut un împrumut la cifra de ordin superior a descăzutului.

6. Se execută instrucţiunea DAA, conţinutul acumulatorului devenind 96H. Aceste cifre se memorează, reprezentînd cifrele cel mai puţin semnificative ale rezultatului.

7. (A) ← 99H.

8. (A) ← (A) + CY = 99H + 0 = 99H.

9. Se calculează complementul faţă de 10 al ultimelor două cifre ale scăzătorului:

$$\begin{array}{rll} (A) = & 1001 & 1001+ \\ 13H = & 1110 & 1101 \\ \hline 1] & 1000 & 0110 \end{array} \quad \begin{array}{l} \text{complementul faţă} \\ \text{de 2 al numărului 13H} \end{array}$$

10. Se adună conţinutul acumulatorului cu cel de-al doilea octet al descăzutului:

$$\begin{array}{rll} (A) = & 1000 & 0110+ \\ 43H = & 0100 & 0011 \\ \hline & 1100 & 1001 \\ & \longrightarrow CY=0 & \longrightarrow AC=0 \end{array}$$

11. Se execută instrucţiunea DAA pentru a corecta rezultatul adunării anterioare, conţinutul acumulatorului devenind egal cu 29H şi CY=1 indicînd faptul că scăderea s-a făcut fără împrumut.

Cînd după scăderea ultimelor două cifre, de pondere maximă, a două numere în cod BCD, indicatorul CY va fi şters, CY=0, atunci rezultatul operaţiei de scădere este un număr negativ în complement faţă de 10, fără cifra de semn.

Exemplu. Dacă se execută scăderea dintre 1257 şi 1626 conform algoritmului descris mai sus se va obţine rezultatul 9631 şi CY=0, diferenţa reprezentînd complementul faţă de 10 al numărului negativ 0369.

Subrutina SUBCD execută, după algoritmul descris mai sus, diferenţa a două numere în cod BCD cu maximum 255 ×2 cifre. Rezultatul operaţiei se obţine în zona de memorie afectată descăzutului. Semnul este indicat de valoarea indicatorului de transport: cînd CY=1 rezultatul este un număr pozitiv, iar cînd CY=0 rezultatul este un număr negativ în complement faţă de 10, fără cifra de semn. Datele de intrare necesare acestei subrutine sînt indicate de conţinutul registrelor: D şi E — adresa zonei de memorie afectată descăzutului; H şi L — adresa zonei de memorie afectată scăzătorului;

B — lungimea celor două zone de memorie în octeți, presupunîndu-se că cele două numere au aceeași lungime. Subrutina alterează conținutul registrelor A, B și al indicatorilor de condiții.

```
SUBCD:  STC             ;POZIȚIONEAZĂ INDICATORUL DE
                        ;TRANSPORT, CY=1
        PUSH  D         ;SALVEAZĂ CONȚINUTUL REG. D ȘI
                        ;E — ADR. DESCĂZUT
        PUSH  H         ;SALVEAZĂ CONȚINUTUL REG. H ȘI
                        ;L — ADR. SCĂZĂTOR
        MVI   A,99H     ;CALCULEAZĂ COMPLEMENTUL FAȚĂ
        ACI   0         ;DE 10 PENTRU DOUĂ CIFRE ALE
        SUB   M         ;SCĂZĂTORULUI
        XCHG            ;SCHIMBĂ CONȚINUTUL REG. D ȘI E CU
                        ;CEL AL REG. H ȘI L
        ADD   M         ;ADUNĂ ÎN COD BCD DOUĂ CIFRE ALE
                        ;DESCĂZUTULUI
        DAA             ;CU COMPLEMENTUL FAȚĂ DE 10 AL
                        ;CIFRELOR CORESPUNZĂTOARE
        MØV   M,A       ;ALE SCĂZĂTORULUI
        XCHG            ;SCHIMBĂ CONȚINUTUL REG. D ȘI E CU
                        ;AL REG. H ȘI L
        INX   H         ;INCREMENTEAZĂ ADRESA
                        ;SCĂZĂTORULUI CU 1
        INX   D         ;INCREMENTEAZĂ ADRESA
                        ;DESCĂZUTULUI CU 1
        DCR   B         ;DECREMENTEAZĂ CU 1 CONTOR
                        ;LUNGIME NUMERE
        JNZ SUBCD+3     ;SALT CÎND CONTORUL ESTE DIFERIT
                        ;DE ZERO
        PØP   H         ;REFACE CONȚINUTUL REG. H ȘI L
        PØP   D         ;REFACE CONȚINUTUL REG. D ȘI E
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
```

Această subrutină nu alterează conținutul zonei de memorie afectată scăzătorului.

5.4.9.5. Înmulțirea a 2 numere

Metodele utilizate la înmulțirea a două numere în cod BCD pot fi clasificate în concordanță cu modul în care sînt reprezentate numerele negative. Pentru numerele reprezentate în cod BCD în mărime și semn există mai multe metode de înmulțire, dintre care amintim:

— metoda adunării repetate;

— metoda dublării și înjumătățirii.

În fiecare dintre aceste metode, semnul produsului este suma modulo 2 a cifrelor de semn ale numerelor care se înmulțesc. Algoritmii de înmulțire ce derivă din aceste metode determină la început semnul rezultatului, după care înmulțirea celor două numere se execută asemănător cu cea a numerelor fără semn. Înmulțind două numere în cod BCD fără semn cu N1 și, respectiv, N2 cifre, se va obține un număr cu N1 +N2 cifre.

Metoda adunării repetate constă în adunări repetate în cod BCD ale deînmulțitului cu el însuși și în deplasarea produsului parțial după înmulțirea cu fiecare cifră a înmulțitorului. Numărul necesar de adunări ale deînmulțitului cu el însuși este egal cu valoarea corespunzătoare a cifrei înmulțitorului. Algoritmul este asemănător metodei de înmulțire utilizate în aritmetica obișnuită. În cursul operației de înmulțire a două numere, fiecare având n cifre, pentru fiecare cifră a înmulțitorului sînt necesare maximum nouă adunări, deci în medie 4,5 adunări ale deînmulțitului cu el însuși.

Metoda dublării și înjumătățirii folosește un algoritm bazat pe următoarele relații matematice:

$$D \cdot I = 2 \cdot D \cdot (I/2) \qquad \text{dacă I este par} \qquad (5.6)$$

$$D \cdot I = 2 \cdot D \cdot ((I - 1)/2) + D \qquad \text{dacă I este impar} \qquad (5.7)$$

în care D și I reprezintă valoarea absolută a deînmulțitului și înmulțitorului în cod BCD.
Înmulțirea se efectuează iterativ prin aplicarea repetată a celor două relații. Acest procedeu este exemplificat în continuare pentru înmulțirea numerelor 36 și 69:

$$\begin{aligned} 36 \times 69 &= 2 \times 36 \times (68/2) + 36 \\ &= 2 \times 72 \times (34/2) + 36 \\ &= 2 \times 144 \times (16/2) + 36 + 144 \\ &= 2 \times 288 \times (8/2) + 36 + 144 \\ &= 2 \times 576 \times (4/2) + 36 + 144 \\ &= 2 \times 1152 \times (2/2) + 36 + 144 = 2304 + 36 + 144 \\ &= 2484. \end{aligned}$$

Desfășurarea operației de înmulțire arătată poate fi pusă sub forma din figura 5.24.

Înmulțitori (I)	*Deînmulțiți* (D)	*Produse-parțiale*
69*	36	36
34	72	
17*	144	144
8	288	
4	576	
2	1152	
1*	2304	2304
0	4608	
		2484

Fig. 5.24. Exemplu de înmulțire a 2 numere zecimale utilizînd metoda dublării și înjumătățirii

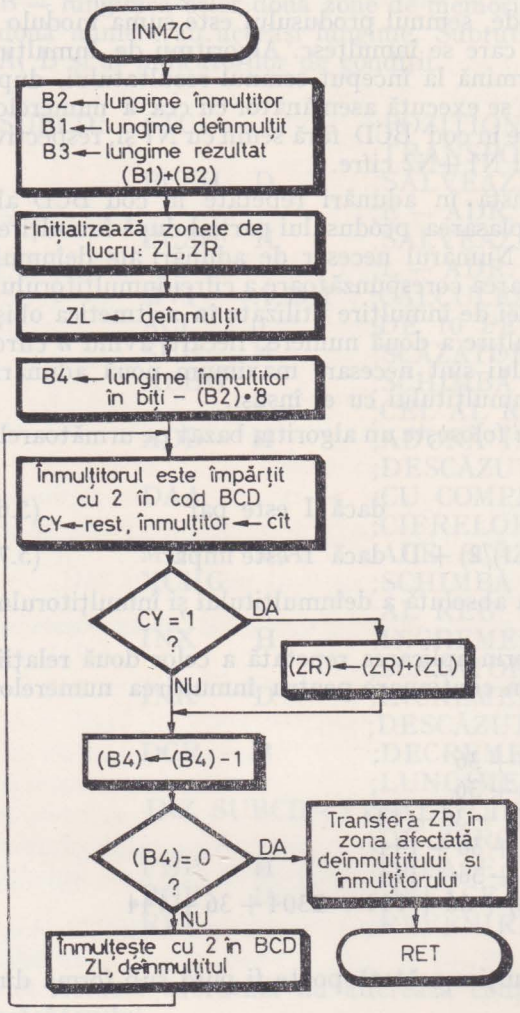


Fig. 5.25. Algoritm de înmulțire a 2 numere în cod BCD fără semn folosind metoda dublării și înjumătățirii

După cum s-a arătat, produsul se obține prin însumarea deînmulțiților cărora le corespund înmulțitori impari, marcați în exemplul din figura 5.24 cu *. Această metodă necesită executarea a trei operații în cod BCD: înmulțire cu 2, împărțire cu 2 și adunare. Algoritmul de înmulțire corespunzător este descris în organigrama din figura 5.25 și în subrutina corespunzătoare acesteia, INMZC, prezentată mai jos.

Modul de executare a împărțirii cu 2 a unui număr zecimal în cod BCD a fost prezentat în § 5.4.9.2.2. iar a operației de adunare a două numere în cod BCD în § 5.4.9.3.

Înmulțirea cu 2 a unui număr zecimal în cod BCD se realizează adunînd numărul cu el însuși în cod BCD. Această operație este efectuată cu ajutorul subrutinei DSBCD, prezentată în continuare. Datele de la intrare necesare sînt conținute în registrele: H și L — adresa zonei unde este memorat numărul în cod BCD și B — lungimea acestuia în octeți. Subrutina alterează conținutul registrelor A, B și al indicatorilor de condiții.

```
DSBCD: XRA  A          ;ȘTERGE INDICATORUL DE TRANSPORT,
                       ;CY=0
       PUSH H          ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
       MØV  A,M        ;TRANSFERĂ 2 CIFRE ALE NR. ÎN
                       ;ACUMULATOR
       ADC  M          ;ÎNMULȚEȘTE CU 2 CONȚINUTUL
       DAA             ;ACUMULATORULUI ÎN COD BCD ȘI
       MØV  M,A        ;MEMOREAZĂ REZULTATUL
       INX  H          ;INCREMENTEAZĂ CU 1 ADRESA
                       ;NUMĂRULUI
       DCR  B          ;DECREMENTEAZĂ CONTOR LUNGIME
                       ;NUMĂR
       JNZ  DSBCD+2    ;SALT CÎND CONTORUL ESTE DIFERIT
                       ;DE 0
       PØP  H          ;REFACE CONȚINUTUL REG. H ȘI L
       RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
```

Menționăm că subrutina de înmulțire INMZC permite înmulțirea a 2 numere zecimale reprezentate în cod BCD fără semn. Pentru a realiza înmulțirea a 2 numere zecimale reprezentate în cod BCD în mărime și semn este necesar ca la începutul programului prezentat să se mai adauge o secvență prin care se calculează semnul rezultatului și se maschează cifrele de semn al celor două numere, la terminarea operației rezultatul fiind înzestrat cu semn.

În subrutina INMZC produsul se obține în zonele de memorie consecutive afectate deînmulțitului și înmulțitorului și poate avea maximum 255×2 cifre, iar înmulțitorul maximum 30×2 cifre. Această subrutină folosește două zone de lucru: ZL pentru depozitarea temporară a deînmulțitului și ZR pentru depozitarea temporară a rezultatului.

Numărul de iterații (bucle) necesare obținerii produsului celor două numere s-a luat egal cu numărul de biți ai înmulțitorului. Pentru a evita un număr suplimentar de iterații, programul principal trebuie să comunice subrutinei INMZC numărul exact (număr cifre $\times$ 2) de octeți ai înmulțitorului.

Un alt procedeu de oprire a iterațiilor este testarea condiției de înmulțitor nul la sfîrșit de buclă. Această metodă este indicată pentru înmulțitori cu un număr mic de cifre.

Datele necesare subrutinei INMZC sînt conținute în registrele: H și L — adresa deînmulțitului; D și E — adresa înmulțitorului; B — lungimea înmulțitorului în octeți; C — lungimea deînmulțitului în octeți. Această subrutină presupune că deînmulțitul și înmulțitorul sînt depozitați în zone de memorie consecutive. Subrutina alterează conținutul registrelor A, B, C, D, E și al indicatorilor de condiții. Adresa rezultatului este indicată de conținutul registrelor H și L, deînmulțitul și înmulțitorul fiind pierduți.

```
INMZC:  PUSH  H      ;SALVEAZĂ ADRESA DEÎNMULŢITULUI
        MØV   A,B    ;TRANSFERĂ ÎN ACUMULATOR
                     ;LUNGIMEA ÎNMULŢITORULUI
        STA   B2     ;MEMOREAZĂ CONŢINUT ACUMULATOR
        ADD   C      ;CALCULEAZĂ LUNGIMEA REZULTA-
                     ;TULUI
        STA   B3     ;MEMOREAZĂ LUNGIMEA REZULTA-
                     ;TULUI
        MØV   A,C    ;TRANSFERĂ ÎN ACUMULATOR LUNGI-
                     ;MEA DEÎNMULŢITULUI
        STA   B1     ;MEMOREAZĂ LUNGIMEA
                     ;DEÎNMULŢITULUI
        PUSH  D      ;SALVEAZĂ ADRESA ÎNMULŢITORULUI
        XCHG         ;SALVEAZĂ ÎN REG. D ŞI E ADRESA
                     ;DEÎNMULŢITULUI
        MVÌ   B,255  ;ÎNCARCĂ ÎN REG. B LUNGIMEA UNEI
                     ;ZONE DE LUCRU
        LXI   H,ZL   ;ÎNCARCĂ ÎN REG. DUBLU H ADRESA
                     ;ZONEI DE LUCRU
        CALL  INMEM  ;ŞTERGE ZONA DE MEMORIE ZL
        MVÌ   B,255  ;ŞTERGE ZONA DE MEMORIE
        LXÌ   H,ZR   ;FOLOSITĂ PENTRU MEMORAREA
                     ;TEMPORARĂ
        CALL  INMEM  ;A REZULTATULUI, ZR
        LDA   B1     ;ÎNCARCĂ ÎN ACUMULATOR
                     ;LUNGIMEA DEÎNMULŢITULUI
        MØV   B,A    ;TRANSFERĂ ÎN REG. B CONŢINUTUL
                     ;ACUMULATORULUI
        LXI   H,ZL   ;ÎNCARCĂ ÎN REG. H ŞI L ADRESA
                     ;ZONEI DE LUCRU, ZL
        CALL  TRANS  ;TRANSFERĂ DEÎNMULŢITUL ÎN ZONA
                     ;ZL
        LDA   B2     ;ÎNCARCĂ ÎN ACUMULATOR LUNGIMEA
                     ;ÎNMULŢITORULUI
        RLC          ;CALCULEAZĂ LUNGIMEA
        RLC          ;ÎNMULŢITORULUI
        RLC          ;ÎN BIŢI
        STA   B4     ;MEMOREAZĂ LUNGIMEA ÎNMULŢITO-
                     ;RULUI, ÎN BIŢI
IMUL1:  PØP   D      ;ADRESA ÎNMULŢITORULUI ÎN REG. D
                     ;ŞI E
        LDA   B2     ;ÎNCARCĂ ÎN REG. A LUNGIMEA
                     ;ÎNMULŢITORULUI
        MØV   C,A    ;TRANSFERĂ ÎN REG. C CONŢINUTUL
                     ;ACUMULATORULUI
        CALL  DDBCD  ;ÎMPARTE ÎNMULŢITORUL CU 2, RESTUL
                     ;ÎN CY
        PUSH  D      ;SALVEAZĂ ADRESA ÎNMULŢITORULUI
```

```
        JNC     IMUL2   ;SALT DACĂ ÎMPĂRȚITORUL ANTERIOR
                        ;A FOST UN NUMĂR PAR, CY=0
        LXI     D,ZR    ;ADRESA REZULTATULUI TEMPORAR
                        ;ÎN REG. D ȘI E
        LXI     H,ZL    ;ADRESA DE LUCRU PENTRU
                        ;DEÎNMULȚIT ÎN REG. H ȘI L
        LDA     B3      ;ÎNCARCĂ ÎN REG. A LUNGIMEA
                        ;REZULTATULUI
        MØV     B,A     ;ÎNCARCĂ ÎN REG. B LUNGIMEA
                        ;REZULTATULUI
        CALL    ADBCD   ;ADUNĂ ÎN BCD, ZR=ZR+ZL
IMUL2:  LDA     B4      ;ÎNCARCĂ ÎN REG. A NR. BIȚI
                        ;ÎNMULȚITOR
        DCR     A       ;S-A TERMINAT OPERAȚIA DE
                        ;ÎNMULȚIRE?
        JZ      IMUL3   ;DA, SALT
        LXI     H,ZL    ;NU, ADRESA DE LUCRU PENTRU
                        ;DEÎNMULȚIT ÎN REG. H,L
        LDA     B3      ;LUNGIMEA REZULTATULUI ÎN
                        ;ACUMULATOR
        MØV     B,A     ;TRANSFERĂ CONȚINUTUL REG. A ÎN
                        ;REG. B
        CALL    DSBCD   ;ÎNMULȚEȘTE ÎN BCD DEÎNMULȚITUL
                        ;CU 2
        JMP     IMUL1   ;RELUARE BUCLĂ
IMUL3:  LXI     D,ZR    ;ÎNCARCĂ ÎN REG. D ȘI E ADR. REZULTAT
                        ;TEMPORAR
        PØP     H       ;REFACE INDICATORUL DE STIVĂ
        PØP     H       ;ADRESA DEÎNMULȚITULUI ÎN REG. H
                        ;ȘI L
        PUSH    H       ;SALVEAZĂ CONȚINUTUL REG. H ȘI L
        LDA     B3      ;ÎNCARCĂ ÎN REG. A NR. OCTEȚI AI
                        ;REZULTATULUI
        MØV     B,A     ;TRANSFERĂ ÎN REG. B CONȚINUTUL
                        ;REG. A
        CALL    TRANS   ;TRANSFERĂ REZULTATUL ÎN ZONA
                        ;AFECTATĂ DEÎNMULȚITULUI ȘI
                        ;ÎNMULȚITORULUI
        PØP     H       ;ÎN HL ADRESA REZULTATULUI
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
INMEM:  MVI     M,0     ;INTRODUCE ZERO ÎNTR-O ZONĂ DE
                        ;MEMORIE, A CĂREI
        INX     H       ;ADRESĂ ESTE INDICATĂ DE CONȚI-
                        ;NUTUL REG. H ȘI L
        DCR     B       ;ȘI LUNGIME, ÎN OCTEȚI, EGALĂ CU
                        ;CONȚINUTUL
        JNZ     INMEM   ;REGISTRULUI B.
        RET
```

```
TRANS:  LDAX   D        ;TRANSFERĂ CONȚINUTUL UNEI ZONE
                        ;DE MEMORIE, A
        MØV    M,A      ;CĂREI ADRESĂ ESTE INDICATĂ DE
                        ;CONȚINUTUL REG.
        INX    D        ;D ȘI E, ÎNTR-O ZONĂ DE MEMORIE
                        ;ADRESATĂ DE
        INX    H        ;CONȚINUTUL REG. H ȘI L; LUNGIMEA
                        ;ACESTEI ZONE
        DCR    B        ;ESTE EGALĂ CU CONȚINUTUL REG. B
        JNZ    TRANS
        RET
; ZONE DE MEMORIE DE LUCRU UTILIZATE DE SUBRUTINA
;INMZC
B1:     DS     1        ;LUNGIMEA DEÎNMULȚITULUI ÎN OCTEȚI
B2:     DS     1        ;LUNGIMEA ÎNMULȚITORULUI ÎN
                        ;OCTEȚI
B3:     DS     1        ;LUNGIMEA REZULTATULUI ÎN OCTEȚI
B4:     DS     1        ;LUNGIMEA ÎNMULȚITORULUI ÎN BIȚI
ZL:     DS     255      ;ZONĂ DE LUCRU PENTRU MEMORAREA
                        ;DEÎNMULȚITULUI
ZR:     DS     255      ;ZONĂ DE LUCRU PENTRU MEMORAREA
                        ;TEMPORARĂ A REZULTATULUI
```

5.4.9.6. Împărțirea a 2 numere

Împărțirea a două numere zecimale reprezentate în cod BCD în mărime și semn necesită următoarele etape:

1. se rețin cifrele de semn ale celor 2 numere în vederea obținerii cifrei de semn a cîtului, care este egală cu suma modulo 2 a acestora;
2. se maschează cifrele de semn ale celor 2 numere;
3. se împart valorile absolute ale celor 2 numere în vederea obținerii cîtului;
4. se poziționează cifra de semn a cîtului obținut.

Metoda prezentată în continuare se referă la împărțirea a două numere zecimale întregi în cod BCD fără semn. Pentru numerele cu semn, se presupune că cifrele de semn au fost deja reținute și mascate într-o secvență anterioară de program, în continuare executîndu-se împărțirea valorilor absolute.

Prin împărțirea a două numere zecimale întregi cu lungimea de LDIMP octeți, LDIMP $\times$ 2 cifre pentru deîmpărțit și, respectiv, LIMP octeți, LIMP $\times$ 2 cifre pentru împărțitor, se obține un cît cu lungimea de LDIMP-LIMP $+1$ octeți și un rest cu lungimea de LIMP octeți.

Exemplu. Prin împărțirea numerelor 99 975 și 174 care au LDIMP $= 3$ și, respectiv, LIMP $= 2$ se obține cîtul 574, reprezentat pe 2 octeți și restul 101, exprimat pe 2 octeți.

$$99\,975 = 174 \times 574 + 101$$

Metoda descrisă aici se bazează pe principiul de împărțire obișnuită a numerelor întregi din aritmetica zecimală. Această metodă poate fi denumită *metoda scăderii repetate și refacerii restului*. Cu ajutorul metodei se obțin cifrele cîtului, începînd cu cea de pondere maximă, prin scăderea repetată în cod BCD a cifrelor împărțitorului dintr-un număr de cifre ale descăzutului egal cu numărul de cifre ale împărțitorului, începînd cu cifrele de pondere maximă. O cifră a cîtului se obține prin scăderea împărțitorului, de m ori, dintr-un grup de cifre ale descăzutului, pînă cînd se produce un împrumut, o diferență negativă. Valoarea cifrei cîtului va fi egală cu $m-1$; cifra cîtului este mai mică cu 1 decît numărul de scăderi. Scăderea a 2 numere zecimale în cod BCD se execută adunînd primul număr cu complementul față de 10 al celui de-al doilea număr (vezi § 5.4.9.4.). Pentru a obține următoarea cifră a cîtului se reface restul, adunîndu-se rezultatul ultimei scăderi cu împrumutul, după aceea se deplasează deînmulțitul cu o cifră zecimală, 4 biți, spre stînga, și se reia algoritmul de mai sus.

Aflarea tuturor cifrelor cîtului și ale restului se face iterativ, prin decrementarea cu 1 după calculul fiecărei cifre, a contorului de lungime a cîtului, inițial egal cu:

$$(\text{LDIMP} - \text{LIMP} + 1) \times 2 \text{ cifre.}$$

Algoritmul de împărțire a două numere zecimale în cod BCD fără semn bazat pe metoda descrisă mai sus este sintetizat în organigrama din figura 5.26. Subrutina corespunzătoare acesteia, IMBCD, va fi prezentată mai jos. IMBCD utilizează 2 octeți de memorie REZ și CSD pentru calculul cifrelor cîtului; REZ este folosit pentru depozitarea temporară a două cifre ale cîtului, CSD arată poziția cifrei cîtului anterior calculate din REZ — stînga sau dreapta.

Cu ajutorul subrutinei IMBCD se pot împărți numere zecimale care au lungimea de maximum 255×2 cifre, suficientă pentru majoritatea operațiilor.

Datele de intrare necesare sînt indicate de conținutul registrelor: D și E — adresa zonei de memorie unde este depozitat deîmpărțitul; H și L — adresa zonei de memorie unde se găsește împărțitorul; B — lungimea în octeți a zonei de memorie afectată împărțitorului; C — lungimea în octeți a zonei de memorie afectată deîmpărțitului. La ieșirea din această subrutină cîtul este memorat în zona afectată deîmpărțitului, cu adresa și lungimea indicate de conținutul registrelor D și E, respectiv B. Restul va fi memorat în zona afectată împărțitorului, a cărei adresă și lungime sînt indicate de conținutul registrelor H și L, respectiv C.

În consecință, această subrutină alterează zonele de memorie afectate deîmpărțitului și împărțitorului, precum și conținutul registrelor A, B, C și al indicatorilor de condiții.

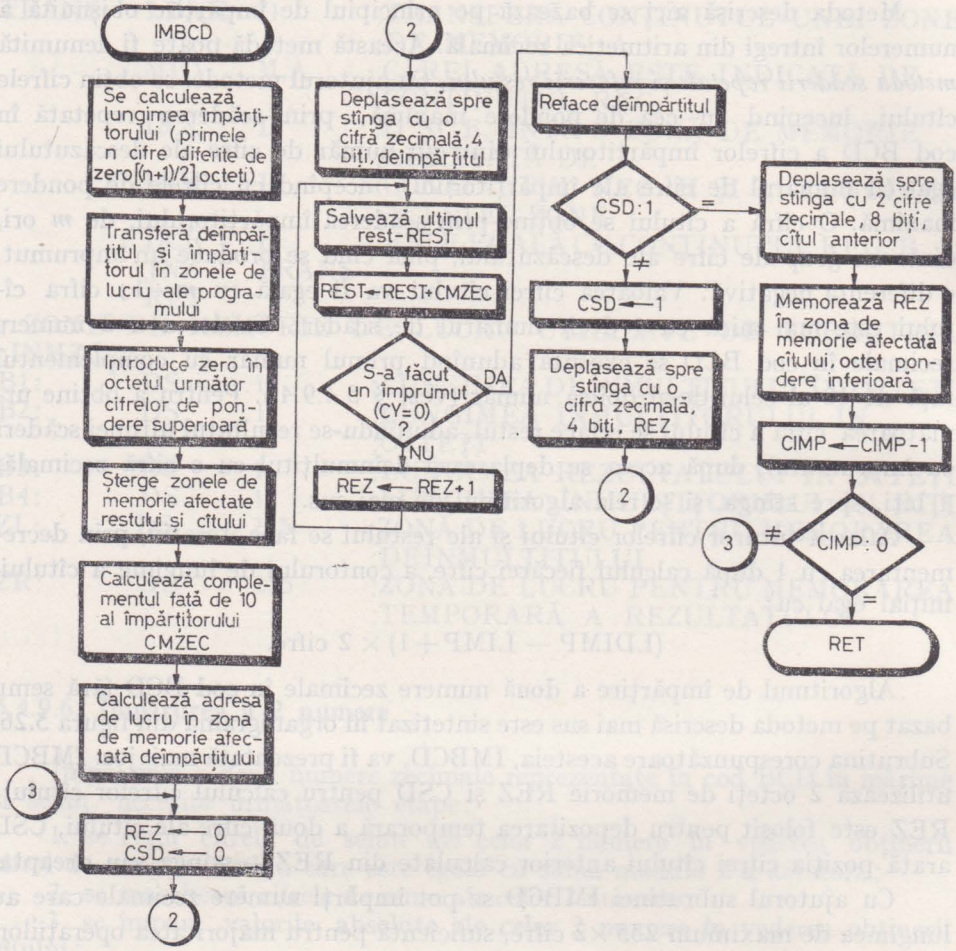


Fig. 5.26. Algoritm de împărțire a 2 numere zecimale în cod BCD fără semn folosind metoda scăderii repetate și refacerii restului parțial.

```
IMBCD:  PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. C
        MØV   C,B    ;TRANSFERĂ ÎN REG. C CONȚINUTUL
                     ;REG. B
        XRA   A      ;ȘTERGE ACUMULATORUL, (A)=0
        MØV   B,A    ;ȘTERGE REG. B, (B)=(A)=0
        DAD   B      ;(H)(L) ← (H)(L)+(B)(C)
        DCX   H      ;DECREMENTEAZĂ CU 1 CONȚINUTUL
                     ;REG. HL
        CMP   M      ;COMPARĂ 2 CIFRE ALE ÎMPĂRȚITORULUI
                     ;CU 0
```

```
JNZ   $+11   ;SALT DACĂ CEL PUȚIN O CIFRĂ ESTE
             ;DIFERITĂ DE 0
DCR   C      ;DECREMENTEAZĂ CU 1 CONTOR
             ;LUNGIME ÎMPĂRȚITOR
JZ    ER     ;SALT DACĂ ÎMPĂRȚITORUL ESTE NUL
JMP   $-9    ;RELUARE TEST CONȚINUT ÎMPĂRȚITOR
MØV   A,C    ;SALVEAZĂ LUNGIMEA ÎMPĂRȚITORULUI
STA   LIMP   ;MEMOREAZĂ LUNGIMEA ÎMPĂRȚITO-
             ;RULUI
PØP   B      ;REFACE CONȚINUTUL REG. C
MØV   B,A    ;TRANSFERĂ LUNGIMEA
             ;ÎMPĂRȚITORULUI ÎN REG. B
PUSH  B      ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
PUSH  D      ;SALVEAZĂ CONȚINUTUL REG. DE —ADR
             ;DEÎMPĂRȚIT
PUSH  H      ;SALVEAZĂ CONȚINUTUL REG. HL—ADR.
             ;ÎMPĂRȚITOR
MØV   A,C    ;TRANSFERĂ ÎN ACUMULATOR LUNGI-
             ;MEA DEÎMPĂRȚITULUI
INR   A      ;INCREMENTEAZĂ CU 1 LUNGIME
             ;DEÎMPĂRȚIT
STA   LDIMP; ;MEMOREAZĂ CONȚINUTUL ACUMULA-
             ;TORULUI
SUB   B      ;CALCULEAZĂ LUNGIMEA REZULTATU-
             ;LUI
STA   LREZ   ;MEMOREAZĂ LUNGIMEA REZULTATU-
             ;LUI
STA   CIMP   ;INIȚIALIZEAZĂ CONTOR ITERAȚII
MØV   B,C;   ;ÎNCARCĂ ÎN REG. B LUNGIME
             ;DEÎMPĂRȚIT
LXI   H,ZDIMP; TRANSFERĂ DEÎMPĂRȚITUL
             ;ÎNTR-O ZONĂ DE
CALL  TRANS  ;MEMORIE DE LUCRU FOLOSITĂ DE
             ;SUBRUTINĂ
MVI   M,0    ;INTRODUCE 0 ÎN LOCAȚIA DE MEMORIE
             ;A CĂREI ADRESĂ ESTE ZDIMP+
             ;(LDIMP) —1
PØP   D      ;ÎNCARCĂ ÎN REG. DE ADRESA ÎMPĂR-
             ;ȚITORULUI
PUSH  D      ;SALVEAZĂ CONȚINUTUL REGISTRELOR
             ;D ȘI E
LDA   LIMP   ;ÎNCARCĂ ÎN REGISTRUL B
MØV   B,A    ;LUNGIMEA ÎMPĂRȚITORULUI
LXI   H,ZIMP ;TRANSFERĂ ÎMPĂRȚITORUL ÎNTR-O
             ;ZONĂ
CALL  TRANS  ;DE MEMORIE DE LUCRU FOLOSITĂ DE
             ;SUBRUTINĂ
PØP   H      ;ÎNCARCĂ ADRESA RESTULUI ÎN REG. HL
```

```
        PØP   D      ;ÎNCARCĂ ADRESA CÎTULUI ÎN REG. DE
        PØP   B      ;REFACE CONȚINUTUL REG. B ȘI C
        PUSH  H      ;SALVEAZĂ ADRESA RESTULUI
        CALL  INMEM  ;ȘTERGE ZONA DE MEMORIE AFECTATĂ
                     ;RESTULUI
        MØV   B,C    ;TRANSFERĂ LUNGIMEA DEÎMPĂRȚI-
                     ;TULUI ÎN REG. B
        XCHG         ;(D)(E) ↔ (H)(L)
        PUSH  H      ;SALVEAZĂ ADRESA CÎTULUI
        CALL  INMEM  ;ȘTERGE ZONA DE MEMORIE AFECTATĂ
                     ;CÎTULUI
        LDA   LIMP   ;ÎNCARCĂ ÎN ACUMULATOR LUNGIMEA
                     ;ÎMPĂRȚITORULUI
        MØV   B,A    ;TRANSFERĂ CONȚINUTUL ACUMULA-
                     ;TORULUI ÎN REG. B
        LXI   H,ZIMP ;ÎNCARCĂ ÎN REG. HL — ADRESA ÎM-
                     ;PĂRȚITORULUI
        MVI   A,99H  ;ÎNCARCĂ ÎN ACUMULATOR 99H
        SUB   M      ;SCADE 2 CIFRE ALE DEÎMPĂRȚITULUI
                     ;DIN 99H
        INR   A      ;INCREMENTEAZĂ CU 1, ÎN COD BCD,
IMP1:   DAA          ;CONȚINUTUL ACUMULATORULUI
        MØV   M,A    ;MEMOREAZĂ COMPLEMENTUL FAȚĂ DE
                     ;10
        PUSH  PSW    ;SALVEAZĂ CONȚINUTUL INDICATORU-
                     ;LUI CY
        INX   H      ;INCREMENTEAZĂ CU 1 ADRESA ÎMPĂR-
                     ;ȚITORULUI
        DCR   B      ;DECREMENTEAZĂ CU 1 CONTOR LUN-
                     ;GIME ÎMPĂRȚITOR
        JZ    IMPX   ;SALT CÎND CONTORUL ESTE ZERO
        MVI   A,99H  ;ÎNCARCĂ ÎN ACUMULATOR 99H
        SUB   M      ;SCADE 2 CIFRE ALE ÎMPĂRȚITORULUI
                     ;DIN 99H
        MØV   M,A    ;SALVEAZĂ CONȚINUTUL ACUMULATO-
                     ;RULUI
        PØP   PSW    ;REFACE CONȚINUTUL INDICATORULUI
                     ;DE TRANSPORT
        MØV   A,M    ;REFACE CONȚINUTUL ACUMULATORU-
                     ;LUI
        ACI   0      ;ADUNĂ CONȚINUTUL ACUMULATORU-
                     ;LUI CU CY
        JMP   IMP1   ;SALT PENTRU CORECȚIA REZULTATU-
                     ;LUI
IMPX:   LDA   LREZ   ;ÎNCARCĂ ÎN ACUMULATOR LUNGIMEA
                     ;REZULTATULUI
        LXI   H,ZDIMP;ÎNCARCĂ ADRESA DEÎMPĂRȚITULUI
                     ;ÎN REG. HL
```

```
        MØV   C,A      ;TRANSFERĂ ÎN REG. C CONȚINUTUL
                       ;ACUMULATORULUI
        MVI   B,0      ;ÎNCARCĂ IMEDIAT 0 ÎN REG. B
        DAD   B        ;(H)(L) ← (H)(L) + (B)(C), ADRESĂ ZONĂ
                       ;DE LUCRU
        PØP   PSW      ;REFACE INDICATORUL DE STIVĂ
        PØP   B        ;ÎNCARCĂ ÎN REG. BC ADRESA CÎTULUI
        PØP   D        ;ÎNCARCĂ ÎN REG. DE ADRESA RESTULUI
        PUSH  B        ;SALVEAZĂ ADRESA CÎTULUI
        PUSH  D        ;SALVEAZĂ ADRESA RESTULUI
        PUSH  H        ;SALVEAZĂ ADRESA ZONEI DE LUCRU DIN
                       ;ZDIMP
ALFA:   XRA   A        ;ȘTERGE CONȚINUTUL ACUMULATORU-
                       ;LUI
        STA   REZ      ;ȘTERGE ZONA DE MEMORIE UTILIZATĂ
                       ;PENTRU CALCULUL UNUI GRUP DE 2
                       ;CIFRE ALE CÎTULUI
        STA   CSD      ;ȘTERGE INDICATOR POZIȚIE CIFRĂ ÎN
                       ;REZ
        LXI   D,ZDIMP  ;ÎNCARCĂ ÎN REG. DE — ADRESA
                       ;DEÎMPĂRȚITULUI
        LDA   LDIM     ;DEPLASEAZĂ SPRE STÎNGA
        MØV   B,A      ;DEÎMPĂRȚITUL CU O CIFRĂ ZECIMALĂ=
        MVI   C,4      ;4 BIȚI PENTRU CALCULUL UNEI
        CALL  DSCIT    ;NOI CIFRE A CÎTULUI
        PØP   D        ;ÎNCARCĂ ÎN REG. DE ADRESA ZONEI
                       ;DE LUCRU DIN ZDIMP
ALFA1:  PØP   H        ;ÎNCARCĂ ÎN REG. HL ADRESA RESTU-
                       ;LUI
        PUSH  H        ;SALVEAZĂ ADRESA RESTULUI
        PUSH  D        ;SALVEAZĂ CONȚINUTUL REG. DE —
                       ;ADR. ZONĂ LUCRU
        LDA   LIMP     ;SALVEAZĂ ULTIMUL REST ÎN
        MØV   B,A      ;ZONA DE MEMORIE ÎN CARE A
        CALL  TRANS    ;FOST MEMORAT INIȚIAL ÎMPĂRȚITO-
                       ;RUL
        PØP   D        ;REFACE CONȚINUTUL REG. DE
        LXI   H,ZIMP   ;ÎNCARCĂ ÎN REG. HL — ADRESA ÎMPĂR-
                       ;ȚITORULUI
        LDA   LIMP     ;ADUNĂ CIFRELE CORESPUNZĂTOARE
                       ;ALE
        MØV   B,A      ;DEÎMPĂRȚITULUI CU COMPLEMENTUL
        CALL  ADBCD    ;FAȚĂ DE 10 AL ÎMPĂRȚITORULUI
        JNC   ALFA2    ;SALT DACĂ S-A FĂCUT UN ÎMPRUMUT
        LDA   REZ      ;INCREMENTEAZĂ CU 1 REZULTATUL
        INR   A        ;TEMPORAR — CARE VA FI VALOAREA
```

```
        STA     REZ     ;UNEI CIFRE ZECIMALE
        JMP     ALFA1   ;RELUAREA BUCLEI DE CALCUL A UNEI
                        ;CIFRE A CÎTULUI
ALFA2:  PØP     H       ;ÎNCARCĂ ÎN REG. HL ADRESA RESTU-
                        ;LUI
        PUSH    H       ;SALVEAZĂ ADRESA RESTULUI
        PUSH    D       ;SALVEAZĂ ADRESA ZONEI DE LUCRU A
                        ;DEÎMPĂRȚITULUI
        LDA     LIMP    ;ÎNCARCĂ ÎN ACUMULATOR LUNGIMEA
                        ;ÎMPĂRȚITORULUI
        MØV     B,A     ;CARE ESTE ȘI LUNGIMEA RESTULUI
        CALL    TRANS   ;TRANSFERĂ ULTIMUL REST ÎN ZONA
                        ;DE LUCRU
        LDA     CSD     ;ÎNCARCĂ ÎN ACUMULATOR CONTOR PO-
                        ;ZIȚIE
        CPI     1       ;CIFRĂ ȘI VERIFICĂ DACĂ ESTE EGAL
                        ;CU 1
        JZ      IMP6    ;SALT DACĂ S-AU OBȚINUT ÎN REZ 2
                        ;CIFRE
        INR     A       ;INTRODUCE 1 ÎN CONTORUL CARE
        STA     CSD     ;INDICĂ POZIȚIA CIFREI CÎTULUI, CAL-
                        ;CULATĂ ANTERIOR ÎN ZONA REZ
        LDA     REZ     ;DEPLASEAZĂ CIFRA CÎTULUI, DEJA
                        ;CALCULATĂ
        RLC             ;CU 4 BIȚI SPRE STÎNGA,
        RLC             ;O CIFRĂ ZECIMALĂ, PENTRU CALCULUL
        RLC             ;URMĂTOAREI CIFRE A CÎTULUI
        RLC             ;ȘI O MEMOREAZĂ
        STA     REZ
        JMP     ALFA+7; RELUARE PENTRU CALCULUL UNEI
                        ;NOI CIFRE A CÎTULUI
IMP6:   PØP     D       ;ÎNCARCĂ ÎN REG. DE — ADRESA ZONEI
                        ;DE LUCRU
        PØP     B       ;ÎNCARCĂ ÎN REG. BC — ADRESA RES-
                        ;TULUI
        PØP     H       ;ÎNCARCĂ ÎN REG. HL — ADRESA CÎ-
                        ;TULUI
        PUSH    B       ;SALVEAZĂ ADRESA RESTULUI
        XCHG            ;DE — ADR. CÎTULUI ȘI HL — ADR. ZO-
                        ;NEI DE LUCRU
        MVI     C,8     ;DEPLASEAZĂ CIFRELE CÎTULUI CU 8
                        ;BIȚI
        LDA     LREZ    ;2 CIFRE ZECIMALE, SPRE STÎNGA
        MØV     B,A     ;PENTRU A PUTEA MEMORA
        CALL    DSCIT   ;ULTIMELE 2 CIFRE CALCULATE
        LDA     REZ     ;ÎNCARCĂ ÎN REG. A CELE 2 CIFRE ALE
                        ;CÎTULUI CALCULATE ANTERIOR
```

```
        XCHG            ;HL-ADR. CÎTULUI ȘI DE-ADRESA ZONEI
                        ;DE LUCRU
        MØV   M,A       ;MEMOREAZĂ ÎN ZONA CÎTULUI ULTI-
                        ;MELE 2 CIFRE CALCULATE
        PØP   B         ;ÎNCARCĂ ÎN REG. BC ADRESA RESTU-
                        ;LUI
        PUSH  H         ;SALVEAZĂ ADRESA CÎTULUI
        PUSH  B         ;SALVEAZĂ ADRESA RESTULUI
        PØP   H         ;ÎNCARCĂ ÎN REG. HL ADRESA RESTULUI
        PUSH  H         ;SALVEAZĂ ADRESA RESTULUI
        PUSH  D         ;SALVEAZĂ ADRESA ZONEI DE LUCRU
                        ;DIN ZDIMP
        LDA   CIMP      ;DECREMENTEAZĂ CU 1 CONTORUL CARE
                        ;INDICĂ
        DCR   A         ;NUMĂRUL DE ITERAȚII CE TREBUIE
                        ;EXECUTATE
        STA   CIMP      ;PENTRU AFLAREA CÎTULUI
        JNZ   IMP6-10   ;SALT CÎND CONTORUL ESTE DIFERIT
                        ;DE ZERO
REVIM:  PØP   B         ;REFACE STIVA
        PØP   H         ;REFACE ADRESA RESTULUI ÎN REG.
                        ;HL
        PØP   D         ;REFACE ADRESA CÎTULUI ÎN REG. DE
        LDA   LREZ      ;ÎNCARCĂ ÎN ACUMULATOR LUNGIMEA
                        ;REZULTATULUI
        MØV   B,A       ;TRANSFERĂ ÎN REG. B CONȚINUTUL
                        ;ACUMULATORULUI
        LDA   LIMP      ;ÎNCARCĂ ÎN ACUMULATOR LUNGIMEA
                        ;RESTULUI
        MØV   C,A       ;TRANSFERĂ ÎN REG. C CONȚINUTUL
                        ;ACUMULATORULUI
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
ER:     PØP   B         ;REFACE STIVA
        ———             ;MESAJ EROARE — ÎMPĂRȚITOR NUL
        RET             ;REVENIRE ÎN PROGRAMUL PRINCIPAL
DSCIT:  PUSH  B         ;SALVEAZĂ CONȚINUTUL REG. B
        CALL  DSBIN     ;DEPLASEAZĂ SPRE STÎNGA CU UN BIT
;CONȚINUTUL UNEI ZONE DE MEMORIE CU ADRESA CONȚINUTĂ
                        ;ÎN REG. DE
        PØP   B         ;REFACE CONȚINUTUL REG. B
        DCR   C         ;DECREMENTEAZĂ CU 1 CONTOR DEPLA-
                        ;SĂRI
        JNZ   DSCIT     ;SALT CÎND CONTORUL ESTE DIFERIT
                        ;DE ZERO
        RET             ;REVENIRE ÎN PROGRAMUL APELANT
```

```
REZ:    DS    1      ;ZONĂ DE MEMORIE UTILIZATĂ PEN-
;TRU MEMORAREA TEMPORARĂ A 2 CIFRE CURENTE ALE CÎTULUI
LIMP:   DS    1      ;ZONĂ TAMPON — LUNGIME ÎMPĂRȚI-
                     ;TOR
LDIMP:  DS    1      ;ZONĂ TAMPON — LUNGIME DEÎMPĂR-
                     ;ȚIT
LREZ:   DS    1      ;ZONĂ TAMPON — LUNGIME CÎT
CIMP:   DS    1      ;ZONĂ TAMPON — CONTOR ITERAȚII
ZIMP:   DS    20     ;ZONĂ PENTRU MEMORAREA TEMPORA-
                     ;RĂ A ÎMPĂRȚITORULUI
ZDIMP:  DS    40     ;ZONĂ PENTRU MEM. TEMPORARĂ A
                     ;DEÎMPĂRȚITULUI
CSD:    DS    1      ;CONTOR POZIȚIE CIFRĂ — STÎNGA SAU
                     ;DREAPTA
```

5.4.9.7. Compararea a 2 numere

Compararea a 2 numere zecimale în cod BCD se execută pe principii asemănătoare cu compararea a 2 numere în cod binar. Numerele zecimale în cod BCD fără semn pot fi comparate folosind un program de comparare a numerelor în cod binar.

Numerele zecimale cu semn reprezentate în cod BCD vor putea fi comparate numai ținînd cont de semnul lor. În figura 5.27 este dată organigrama

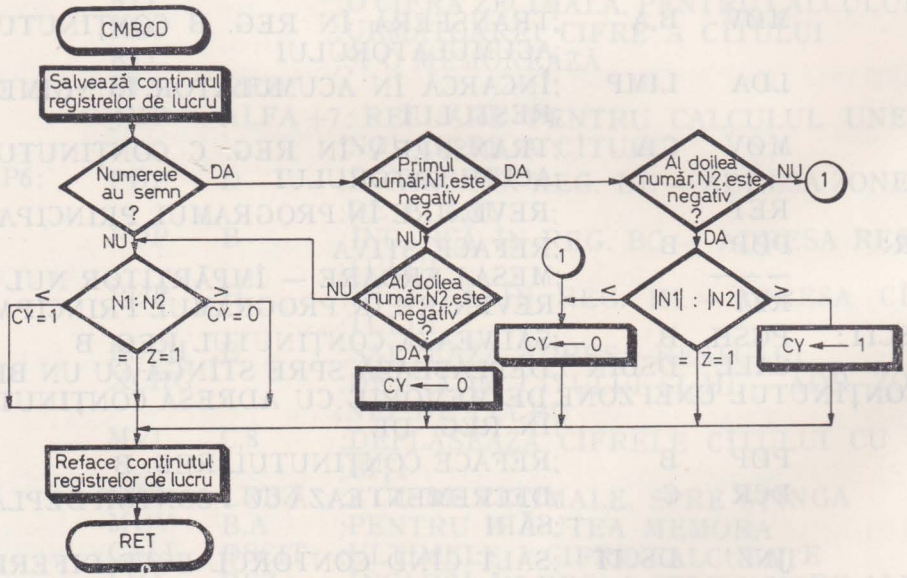


Fig. 5.27. Algoritm de comparare a 2 numere zecimale în cod BCD în mărime și semn sau în cod BCD fără semn.

generală de comparare a două numere zecimale exprimate în cod BCD fără semn sau în cod BCD în mărime și semn. Pentru ca programul corespunzător acestui algoritm, CMBCD, prezentat mai jos, să fie înștiințat dacă cele două numere au semn sau nu, trebuie ca la intrare să se indice acest lucru prin intermediul registrului C.

Datele de intrare necesare subrutinei CMBCD sînt indicate de conținutul registrelor: D și E — adresa zonei de memorie unde se află primul număr; H și L — adresa zonei de memorie unde se află cel de-al doilea număr; B — lungimea celor două numere în octeți; C — indicatorul de semn. Conținutul registrului C egal cu 0 va indica faptul că cele două numere în cod BCD sînt fără semn. Dacă C este egal cu 1 sau, în general, diferit de 0 cele două numere zecimale sînt reprezentate în cod BCD în mărime și semn.

La ieșirea din subrutină indicatorii de condiții vor fi poziționați în funcție de mărimea și semnul numerelor comparate după cum urmează:

1. CY=1 dacă primul număr este mai mic decît al doilea;
2. Z=1 dacă cele două numere sînt egale;
3. CY=0 dacă primul număr este mai mare.

În programul principal valoarea indicatorilor este testată în ordinea prezentată mai sus; valoarea celorlalți indicatori nu prezintă interes aici.

Conținutul celor 2 zone de memorie, precum și al registrelor folosite ca date de intrare nu este alterat la ieșirea din subrutină.

```
CMBCD:  PUSH  H       ;SALVEAZĂ ADRESA CELUI DE-AL DOI-
                      ;LEA NUMĂR
        PUSH  D       ;SALVEAZĂ ADRESA PRIMULUI NUMĂR
        PUSH  B       ;SALVEAZĂ LUNGIMEA NUMERELOR ȘI
                      ;INDICATORUL DE SEMN
        MØV   C,B     ;TRANSFERĂ LUNGIME NUMERE ÎN REG.
                      ;C
        MVI   B,0     ;INTRODUCE 0 ÎN REG. B
        DCX   B
        DAD   B       ;(H)(L) ← (H)(L) + (B)(C), ADRESA OCTE-
                      ;TULUI DE PONDERE MAXIMĂ A PRI-
                      ;MULUI NUMĂR
        XCHG          ;(H)(L) ↔ (D)(E)
        DAD   B       ;(H)(L) ← (H)(L) + (B)(C), ADRESA OCTETU-
                      ;LUI DE PONDERE MAXIMĂ A CELUI
                      ;DE-AL DOILEA NUMĂR
        XCHG          ;(H)(L) ↔ (D)(E)
        PØP   B       ;REFACE LUNGIMEA NUMERELOR ȘI
                      ;INDICATORUL DE SEMN
        PUSH  B       ;SALVEAZĂ CONȚINUTUL REG. B ȘI C
        MØV   A,C     ;TRANSFERĂ ÎN ACUMULTOR INDICA-
                      ;TORUL DE SEMN
        CPI   0       ;CELE 2 NUMERE AU SEMN ?
        JNZ   CSEMN   ;SALT DACA SÎNT CU SEMN
```

```
FSEMN:  LDAX   D      ;ÎNCARCĂ ÎN REG. A UN OCTET AL
                      ;PRIMULUI NUMĂR
        CMP    M      ;COMPARĂ (A) CU OCTETUL CORESPUN-
                      ;ZĂTOR AL CELUI DE-AL DOILEA NU-
                      ;MĂR
        JC     REV    ;SALT DACĂ (A) < ((H)(L))
        JNZ    REV    ;SALT DACĂ (A) > (H)(L))
        DCX    H      ;DECREMENTEAZĂ CU 1 ADRESA CELUI
                      ;DE-AL DOILEA NR.
        DCX    D      ;DECREMENTEAZĂ CU 1 ADRESA PRI-
                      ;MULUI NUMĂR
        DCR    B      ;DECREMENTEAZĂ CU 1 LUNGIMEA CE-
                      ;LOR 2 NUMERE
        JNZ    FSEMN  ;SE REIA OPERAȚIA DE COMPARAȚIE
                      ;DACĂ (B) = 0
REV:    PØP    B      ;REFACE CONȚINUTUL REGISTRULUI
                      ;DUBLU B
        PØP    D      ;REFACE CONȚINUTUL REGISTRULUI
                      ;DUBLU D
        PØP    H      ;REFACE CONȚINUTUL REGISTRULUI
                      ;DUBLU H
        RET           ;REVENIRE PROGRAMUL APELANT
CSEMN:  LDAX   D      ;ÎNCARCĂ ÎN REG. A OCTETUL DE PON-
                      ;DERE MAXIMĂ
        RAL           ;TESTEAZĂ CIFRA DE SEMN A PRIMU-
                      ;LUI NUMĂR
        JC     NEG    ;SALT DACĂ NUMĂRUL ESTE NEGATIV
        MØV    A,M    ;ÎNCARCĂ ÎN REG. A OCTETUL DE PON-
                      ;DERE MAXIMĂ
        RAL           ;AL CELUI DE-AL DOILEA NUMĂR ȘI
                      ;TESTEAZĂ SEMNUL
        JNC    FSEMN  ;CELE 2 NUMERE SÎNT POZITIVE
REVI:   STC           ;POZIȚIONEAZĂ INDICATORUL CARRY,
                      ;CY=1
        CMC           ;CY=0, PRIMUL NUMĂR ESTE NEGATIV
                      ;ȘI AL
        JMP    REV    ;DOILEA POZITIV
NEG:    MØV    A,M    ;ÎNCARCĂ ÎN REG. A OCTETUL DE PON-
                      ;DERE MAXIMĂ
        RAL           ;AL CELUI DE-AL DOILEA NUMĂR ȘI
                      ;TESTEAZĂ SEMNUL
        JC     TCSEM  ;SALT DACĂ NUMĂRUL ESTE NEGATIV
        STC           ;CY=1, PRIMUL NUMĂR ESTE POZITIV
                      ;ȘI CEL
        JMP    REV    ;DE-AL DOILEA NEGATIV
```

```
TCSEM:  LDAX  D      ;COMPARĂ 2 NUMERE NEGATIVE
        CMP   M
        JC    REV1   ;SALT DACĂ |(A)| < |((H)(L))|
        JNZ   TCSEM-4 ;SALT DACĂ |(A)| > |((H)(L))|
        DCX   H      ;DECREMENTEAZĂ CU 1 ADRESA CELUI
                     ;DE-AL DOILEA NR.
        DCX   D      ;DECREMENTEAZĂ CU 1 ADRESA PRIMU-
                     ;LUI NUMĂR
        DCR   B      ;DECREMENTEAZĂ CU 1 CONTORUL DE
                     ;LUNGIME
        JNZ   TCSEM  ;SE REIA OPERAȚIA DE COMPARARE
                     ;DACĂ (B)=0
        JMP   REV    ;S-A TERMINAT OPERAȚIA DE COMPA-
                     ;RARE
```

BIBLIOGRAFIE

1. * * * *Intel 8080 Assembly Language Programming Manual*, September, 1975.
2. * * * *Intel 8080 Microcomputer System User's Manual*, September, 1975.
3. LEVENTHAL, L.A., *8080A/8085A Assembly Language Programming*, Osborne Associates, Inc., Berkeley, California.
4. PEATMAN, J.B., *Microcomputer-based Design*, McGraw Hill, New York, 1977.
5. DANCEA, I., *Microprocesoare — Arhitectură internă, programare, aplicații*, Editura Dacia, Cluj-Napoca, 1979.
6. DONOVAN, J.J., *Systems Programming*, McGraw Hill, Inc., 1972.
7. KNUTH, D.E., *Tratat de programarea calculatoarelor — algoritmi fundamentali*, Editura tehnică, București, 1974.
8. RUS, T., *Structuri de date și sisteme operative*, Editura Academiei RSR, București, 1974.
9. CHU, Y., *Bazele proiectării calculatoarelor numerice*, Editura tehnică, București, 1968, p. 80—102.
10. LARSEN, D.G.; TITUS, J.A.; RONY, P.R., *Microcomputer Interfacing: How Does a Microcomputer Make a Decision?*, Computer Design, 1976, 15, 8, p. 118—120.
11. VITTERA, J.F., *Handling multilevel subroutines and interrupts in microcomputers*, Computer Design, 1978, 17, 1, p. 109—115.
12. HERTZ, W., *Indexed addressing for microcomputer*, Computer Design, 1979, 17, 1, p. 99—104.
13. COLLINS, D.C.; GAREN, E.R.; LAZAR, L., *Motorola's 6800 vs Intel's 8080, a side-by-side comparison*, Integrated Computer Systems, Inc., September, 1977, Section 6.
14. SOUCEK, B., *Microprocessors and Microcomputers*, John Wiley & Sons, Inc., 1976.
15. * * * *Pro-Log, Microprocessor user's guide — 1979/1980*, Pro-Log, Corporation, July, 1979.
16. MORSE, S.P.; RAVENEL, B.W.; MAZOR, S.; POHLMAN, W.P., Intel Corporation, *Intel Microprocessors — 8008 to 8086*, Computer, October, 1980.
17. DANAGHEY, L.F.; BOBBA, G.M.; RUBIN, X.K., *Decision — making with flags in process control*, Computer Design, 1976, 15, 12.
18. ȚEPELEA, V.; KOVACS, A; PURICE, E., *SD-8080, un sistem de dezvoltare pentru microprocesoare*, comunicare la sesiunea „ITC — 10 ani de activitate", București, 23—25 noiembrie, 1978.

APLICAŢII ALE MICROPROCESORULUI 8080

6.1. INTERFAŢA PENTRU CONSOLĂ-SERIE

Vom prezenta aici detaliile de proiectare hardware şi de programare ale interfeţei pentru consolă din cadrul SD-8080. Consola este cuplată la SD-8080 printr-o interfaţă-serie în buclă de 20 mA sau în niveluri de tensiune, *full-duplex*, aceasta însemnînd că informaţiile pot tranzita simultan de la calculator spre consolă şi reciproc. Datele care se transmit la sau de la SD-8080 sînt transformate de interfaţă din formă paralelă pe 8 biţi (cod ASCII), în formă serie sau invers. În cazul formatului-serie asincron de transmisie prezentat în figura 6.1 linia de transmisie se găseşte în repaus în starea „1", care se traduce fie prin prezenţa curentului de 20 mA (MARK), fie printr-un nivel de tensiune negativ cuprins între —3 V şi —12 V, în cazul interfeţei în tensiune conformă cu normele CCITT V24. Pentru a semnala receptorului faptul că începe transmisia unui caracter, emiţătorul generează pe durata unui bit, acest interval fiind o constantă a transmisiei, un „0", numit bit de START, permiţînd sincronizarea receptorului. Urmează pe durata a 8 tacte biţii de date, bitul cel mai puţin semnificativ al codului paralel B_0 fiind transmis primul şi bitul cel mai semnificativ B_7 ultimul. Pentru a readuce linia în stare de repaus, creînd condiţiile de resincronizare a receptorului pentru transmisia caracterului următor, emiţătorul adaugă biţilor de informaţie două tacte de inactivitate, în care linia este la „1", reprezentînd cei doi biţi de STOP din figura 6.1. Nivelul electric asociat lui „0" este absenţa curentului de 20 mA (SPACE), sau în cazul interfeţei în tensiune un nivel cuprins între +3 V şi +12 V. Observăm că în situaţia formatului asincron de transmitere infor-

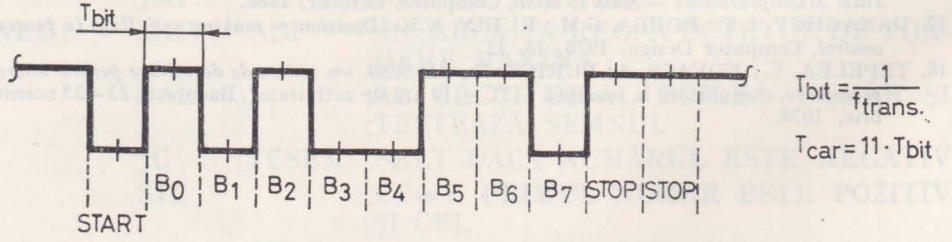


Fig. 6.1. Formatul asincron de transmitere-serie a datelor

mația de sincronizare între emițător și receptor, constituită din cei 3 biți suplimentari, unul de START și doi de STOP, însoțește fiecare caracter în parte, de unde și numele de format 8/11. Acest raport reprezintă o măsură a eficienței transmisiei, deoarece informația de sincronizare este nepurtătoare de informație. Fiind mai puțin eficient, sistemul asincron este folosit pentru comunicații de viteză redusă, specifice legăturilor om-mașină, avînd frecvența de transmisie $f_{trans} \leqslant 19\,200$ biți/s (Bauds). Pentru transmisii de viteze mai mari, specifice legăturilor calculator-calculator, se folosește formatul sincron, care este mai eficient din punctul de vedere al raportului între informația utilă și cea de sincronizare, întrucît aceasta din urmă nu însoțește fiecare caracter în parte, ci fiecare bloc de caractere de date (mesaj). După cum se poate vedea din tabelele 4.3 și 4.4 interfața pentru consolă folosește 4 adrese de I/E enumerate mai jos:

1. Adresa de ieșire FD_{16}; executarea unei instrucțiuni ØUT 0FDH, conținutul acumulatorului avînd bitul 0 poziționat, va produce avansul cu un pas al benzii perforate montate în cititorul de bandă al consolei, în acest caz un Teletype ASR 33 (TTY) lucrînd pe viteza de 110 Bd. Caracterul citit de pe bandă este serializat în interiorul TTY la fel ca și caracterele tastate de la claviatură și este preluat de sistem cu ajutorul deserializorului interfeței pentru consolă.

2. Adresa de intrare FD_{16}; executarea unei instrucțiuni IN 0FDH încarcă în acumulator un octet de stare a sistemului de I/E din care numai biții 0 și 1 sînt semnificativi în cazul interfeței pentru consolă și anume:

— bitul 0, TXRDY = TRANSMITTER READY, cînd este „1" arată că emițătorul, circuitul de serializare al interfeței, este gata ca la primirea unui caracter de date de la procesor să-l transmită către consolă, unde va fi tipărit;

— bitul 1, RXRDY = RECEIVER READY, cînd este „1" arată că receptorul, circuitul de deserializare al interfeței, a terminat recepționarea tuturor biților de date ai unui caracter transmis de la claviatura consolei, eventual de la cititorul de bandă în cazul TTY, caractererul respectiv putînd fi din acest moment preluat de către procesor.

3. Adresa de ieșire FE_{16}; executarea unei instrucțiuni ØUT 0FEH va produce activarea circuitului de serializare al interfeței și tipărirea la consolă a caracterului al cărui cod ASCII s-a aflat în acumulator în momentul executării instrucțiunii.

4. Adresa de intrare FE_{16}; executarea unei instrucțiuni IN 0FEH, atunci cînd bitul RXRDY = „1", produce preluarea în acumulator a codului ASCII al caracterului anterior primit de către deserializor de la consolă.

Pentru cazul TTY viteza de transmitere este de 110 Bd, corespunzător unor constante: $f_{trans} = 110$ Hz, $T_{bit} = 9{,}09$ ms, $T_{car} = 100$ ms (fig. 6.1), rata de transfer fiind de 10 car/s.

Schema circuitului de comandă a cititorului de bandă este prezentată în figura 6.2. Executarea instrucțiunilor:

```
MVI   A,1; POZIȚIONEAZĂ BITUL 0 ÎN ACUMULATOR
ØUT 0FDH; ACȚIONEAZĂ RELEUL DIN TTY
```

produce apariția pe timpul lui I/OW (500 ns) a semnalului $\overline{TRC}$ = „0".

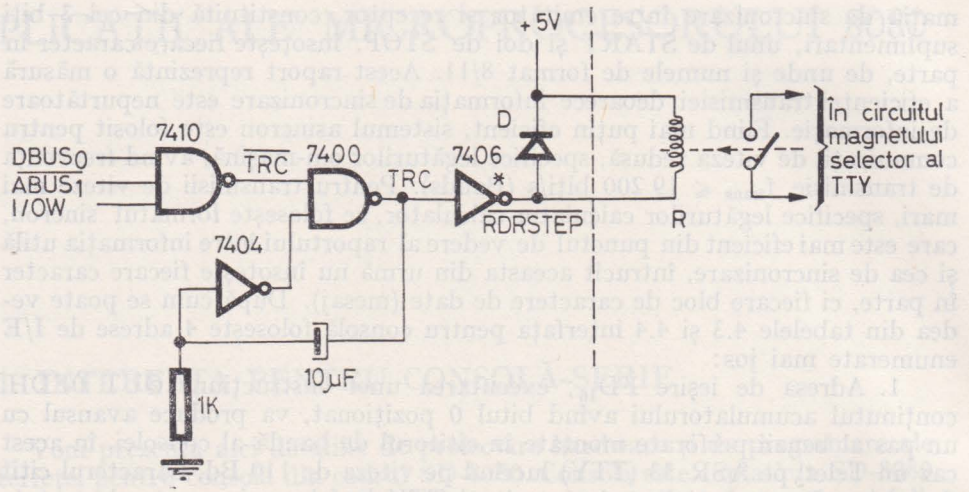


Fig. 6.2. Comanda cititorului de bandă al TTY

Cauzele apariției lui $\overline{TRC}$ = „0" sînt:

— $DBUS_0$ = „1", deoarece s-a încărcat 1 în acumulator înaintea executării instrucțiunii ØUT;

— $\overline{ABUS_1}$ = „1", în timpul instrucțiunii ØUT 0FDH, folosindu-se „selecția liniară" descrisă la § 4.2.5.2;

— I/OW = „1", în timpul stării T_3 a ciclului de IEȘIRE din cadrul instrucțiunii ØUT 0FDH.

Se declanșează astfel monostabilul TRC, care produce un impuls cu durata mai mică de 100 ms (timpul de transmitere a unui caracter). Semnalul $\overline{RDRSTEP}$ = „0", prezent pe durata impulsului TRC, acționează bobina releului R montat în TTY, al cărui contact normal deschis este înseriat în circuitul de alimentare a magnetului selector, care, la rîndul lui, comandă mecanismul de antrenare al cititorului de bandă al TTY. Dioda montată în paralel pe releu are rolul de a suprima tensiunea de autoinducție, care apare la tăierea curentului prin bobină, datorată tranziției la „1" a semnalului $\overline{RDRSTEP}$, echivalentă cu blocarea tranzitorului de ieșire al porții cu „colector în gol" ce produce acest semnal.

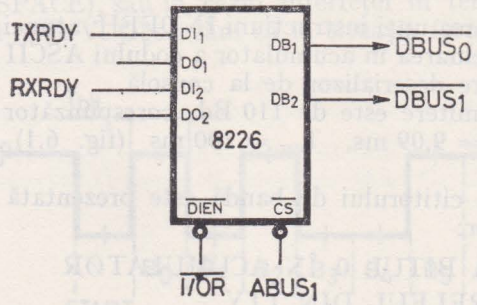


Fig. 6.3. Transmiterea biților de stare TXRDY și RXRDY

Transmiterea biților de stare a interfeței, TXRDY și RXRDY, respectiv pe liniile $\overline{DBUS_0}$ și $\overline{DBUS_1}$ ale magistralei de date se face cu ajutorul circuitului cu ieșire „3 — stări" de tip 8226 prezentat în figura 6.3.

Pentru încărcarea în biții 0 și 1 ai acumulatorului a valorilor lui TXRDY și RXRDY se execută instrucțiunea IN 0FDH; în timpul ciclului de INTRARE al acestei instrucțiuni avem simultan: $\overline{I/OR}$ = „0" și $\overline{ABUS_1}$ = „0" întrunindu-se condițiile pentru selectarea circuitului 8226.

Interfața serializorului cu procesorul este realizată cu ajutorul unui circuit 8212 conectat în modul de ieșire (fig. 6.4*a*) care transmite datele de pe magistrala $\overline{DBUS_i}$ (i = 0—7) pe liniile de încărcare paralelă ale registrului de deplasare ce constituie serializorul, TTO_0—TTO_7. Selectarea acestui circuit are loc în timpul executării instrucțiunii ØUT 0FEH pe baza apariției semnalelor $ABUS_0$ = „0" și I/OW = „1", lucru care produce atît repetarea liniilor magistralei de date pe intrările TTO_i ale serializorului, cît și apariția unui impuls negativ de 500 ns în timpul selectării pe linia $\overline{GOSEND}$ care șterge bistabilul TXRDY, semnalînd în acest fel procesorului că serializorul a fost ocupat. În figura 6.4*b* este prezentat registrul de deplasare de 9 biți ce constituie serializorul. În timpul semnalului END = „1", furnizat de logica de comandă a interfeței, el este încărcat paralel cu codul primit de pe magistrala de date a sistemului, constituind de fapt codul negat al caracterului trimis din acumulator cu ajutorul instrucțiunii ØUT 0FEH, precedat la dreapta de bitul START cu valoare negată, „1". Logica de comandă va înlătura acum semnalul END și va trimite 11 impulsuri de ceas SCLK, producînd deplasarea spre dreapta a conținutului registrului și obținerea la ieșirea sa negată, TTYOUT, a bitului de START, urmat de cei 8 biți informaționali și de cei doi biți de STOP introduși prin stînga registrului în formă negată, „0". Implementarea practică a registrului de deplasare se face cu circuite 7495.

Așa cum se vede în figura 6.4*c*, logica de comandă a serializorului este compusă din două numărătoare de stări în cascadă. Primul numărător funcționează pe baza ceasului SERCLK furnizat de generatorul vitezelor de teletransmisie și avînd o frecvență de 7 ori mai mare ca frecvența de transmisie. De exemplu, pentru viteza de 110 Bd frecvența lui SERCLK este 7 × 110 = = 770 Hz. Notăm că acest ceas este comun pentru serializorul și deserializorul interfeței și că schimbarea frecvenței lui prin reprogramarea generatorului vitezelor de teletransmisie permite schimbarea vitezei de lucru a interfeței pentru consolă la 200, 300, 600 sau 1 200 Bd.

Organigrama de funcționare a primului numărător de stări e prezentată în figura 6.4*d*. Observăm că numărătorul e blocat în starea **A**, pînă la apariția lui TXRDY = „0", ca urmare a executării unei instrucțiuni ØUT 0FEH de către microprocesor. Odată primită comanda de pornire a serializorului, numărătorul se deblochează și parcurge în mod ciclic cele 7 stări, furnizînd la fiecare trecere prin starea **B** ceasul SCLK ce realizează deplasarea la dreapta a serializorului și acționează numărătorul de stări modulo 11.

La al 11-lea semnal de ceas furnizat, după transmiterea tuturor biților (fig. 6.4*e*), numărătorul de stări modulo 11 se reîntoarce în starea sa de repaus **A** în care generează semnalul END, după cum rezultă din organigrama funcționării acestui numărător din figura 6.4*e*. Pe baza apariției lui END numărătorul modulo 7 pune la „1", în starea **D**, bistabilul TXRDY, prin intermediul semnalului S/TXRDY, rămînînd apoi blocat în starea **A**. Celălalt numă-

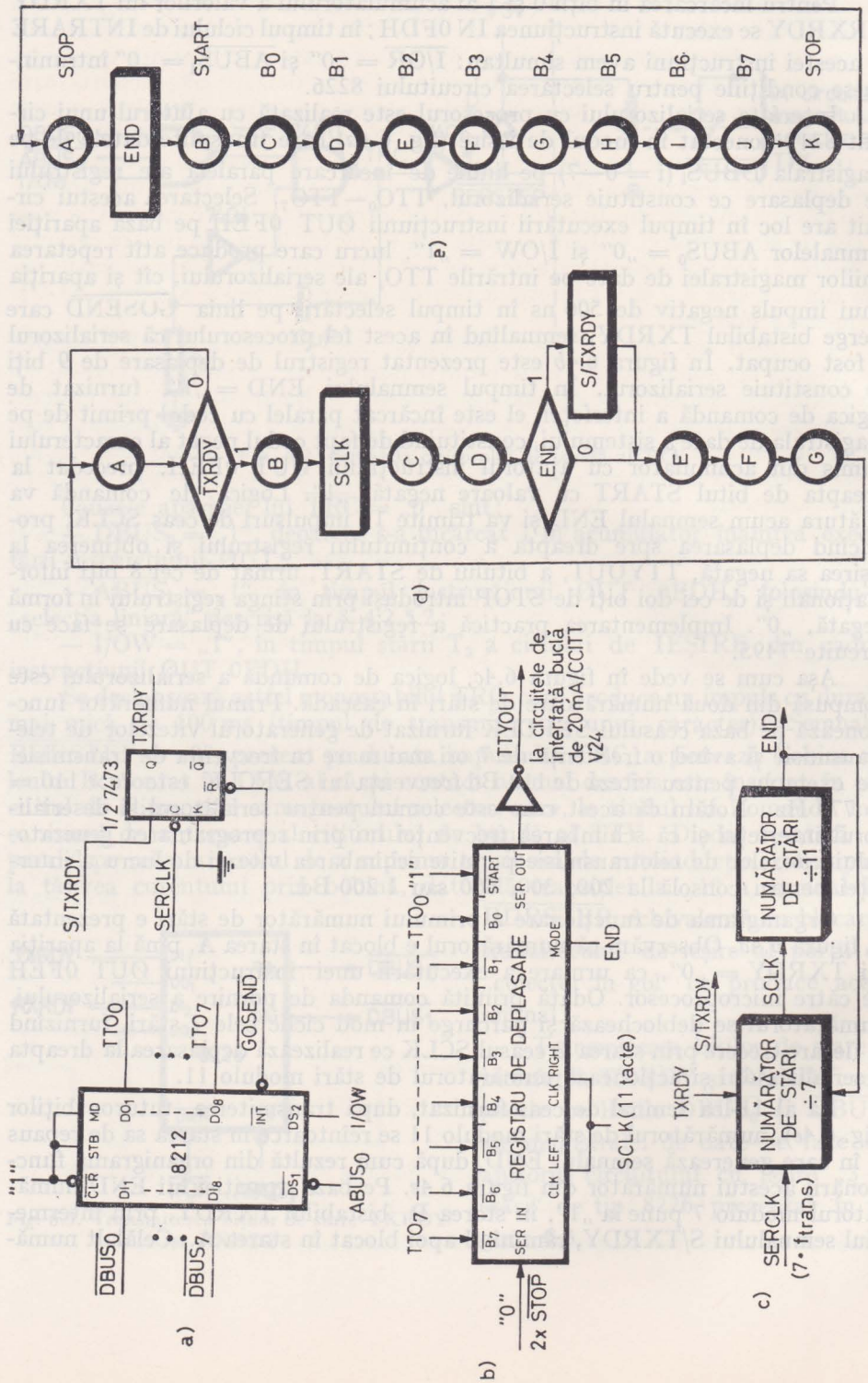


Fig. 6.4. Schemele și organigramele de funcționare ale serializorului interfeței pentru consolă

rător e de asemenea blocat în starea **A** din cauza dispariției ceasului SCLK. Prezența semnalului END în starea de repaus a serializorului creează condițiile pentru memorarea imediată în registrul de deplasare a datelor trimise în paralel pe magistrala de date a sistemului în timpul executării instrucțiunii ØUT 0FEH. Implementarea practică a numărătoarelor de stări și generarea semnalelor de comandă se poate face cu un numărător sincron convenabil ales sau cu bistabili.

Deserializorul interfeței este conectat la magistrala de date a sistemului prin intermediul unui circuit 8212 conectat în modul de intrare (MD = „0"), figura 6.5*a*. El memorează datele de pe liniile TTI_7—TTI_0, constituind codul caracterului recepționat, pe frontul negativ al semnalului STB furnizat de logica de comandă a deserializorului. În același timp semnalul RXRDY, ieșirea $\overline{INT}$ a lui 8212 negată, devine „1" indicînd procesorului că interfața a terminat de recepționat un caracter; acesta poate fi din acest moment preluat în acumulator prin executarea unei instrucțiuni IN 0FEH. În timpul ciclului de INTRARE (I/OR) al instrucțiunii se creează condițiile necesare pentru selectarea circuitului 8212, $\overline{I/OR}$ = „0" și $ABUS_0$ = „0", codul memorat în circuit fiind transpus pe liniile magistralei de date, iar linia RXRDY redevenind „0" ca urmare a ștergerii bistabilului „Service Request FF" din 8212.

Elementul cheie al deserializorului este un registru de deplasare de 8 biți cu intrare-serie TTYIN și ieșire-paralelă TTI_7—TTI_0 acționat cu ceasul RCLK (fig. 6.5*b*). Ca și în cazul serializorului, codul conținut în deserializor este de fapt codul negat al caracterului recepționat, ceea ce se datorează introducerii inversorului de la intrarea registrului de deplasare pe linia TTYIN provenind de la consolă. Acest lucru este necesar pentru a compensa negarea codului introdusă de transferul său pe magistrala de date realizată cu circuite inversoare 8226. Ceasul RCLK are aceeași cadență cu biții recepționați, cu precizarea că logica de comandă asigură ca tranziția activă a lui RCLK să cadă, cu o anumită toleranță, în mijlocul bitului, pentru a asigura o probabilitate minimă de eroare.

Logica de comandă, prezentată în figura 6.5*c* este constituită din două numărătoare de stări cascadate. Primul numărător, modulo 7, funcționează pe baza ceasului comun SERCLK cu frecvența $7 \times f_{transmisie}$. Organigrama lui de funcționare este dată în figura 6.5*d*. El furnizează cele 9 impulsuri de ceas RCLK necesare registrului de deplasare, aceleași impulsuri acționînd și numărătorul modulo 9 care ține evidența biților recepționați (fig. 6.5*e*). După cum observăm pe organigrame, cele două numărătoare se găsesc în repaus în stările **C** pentru numărătorul modulo 7 și **I** pentru numărătorul modulo 9, care furnizează în această stare semnalul ENDR, semnificînd terminarea recepției unui caracter. Condiția de deblocare a numărătorului modulo 7 este să se detecteze simultan RXRDY = „0" și TTYIN = „0", prima condiție semnificînd că procesorul a preluat caracterul anterior, ștergînd astfel bistabilul „Service Request FF" al circuitului 8212 aferent deserializorului, iar cea de-a doua că pe linia de transmisie a datelor se primește un „0" cu semnificația unui bit de START al unui nou caracter. Impulsul de ceas RCLK este generat în cea de-a patra stare de la detectarea bitului de START, mo-

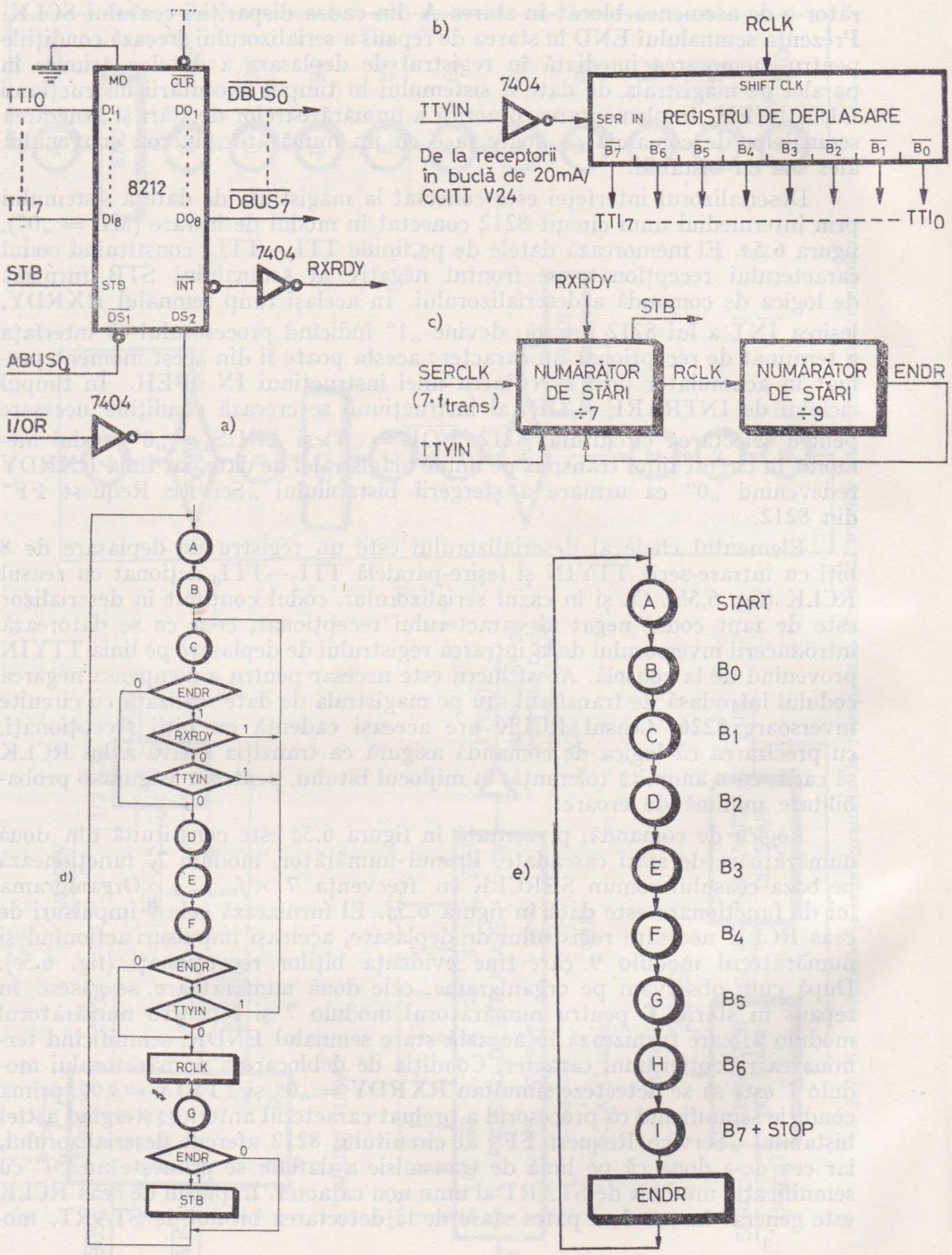


Fig. 6.5. Schemele și organigramele de funcționare ale deserializorului interfeței pentru consolă

ment care coincide cu o aproximație de plus sau minus o jumătate de perioadă a ceasului SERCLK cu centrul bitului de START, deoarece $T_{bit} = 7 \cdot T_{SERCLK}$. Eroarea maximă de sincronizare față de centrul bitului a acestui circuit este de $\pm \frac{1}{2} \cdot \frac{1}{7} T_{bit} = \pm \frac{1}{14} T_{bit}$, adică o eroare de $\pm 7\%$. După sincronizarea pe centrul bitului de START, numărătorul funcționează ciclic furnizînd ceasul RCLK la fiecare a 7-a perioadă a lui SERCLK, deci $T_{RCLK} = T_{bit}$, impulsurile RCLK păstrînd centrarea inițială pentru fiecare bit de date următor. Observăm că în starea **F** a numărătorului modulo 7, în care se generează RCLK, se efectuează un test suplimentar, menit să rejecteze evenutalele impulsuri de zgomot pe linia de transmisie care au o durată mai mică decît $\frac{1}{2} T_{bit}$ și care pot fi confundate cu un bit de START. În adevăr sincronizarea se pierde, numărătorul revenind în starea de așteptare **C**, dacă în starea **F** se detectează ENDR · TTYIN = „1", condiție care ne arată că linia TTYIN a revenit la „1" după aproximativ $\frac{1}{2} T_{bit}$ de la decelarea condiției de sincronizare, deoarece ENDR continuă să fie „1". După apariția primului impuls RCLK numărătorul modulo 9 avansează în starea sa **A**, arătînd că ne aflăm în cursul recepției bitului de START, iar linia ENDR = „0", Numărătorul modulo 7 funcționează în continuare ciclic, pentru cele 8 treceri ulterioare ale sale prin starea **C** testul condiției de sincronizare este eliminat pe baza lui ENDR = „0". După furnizarea celui de-al 9-lea impuls de ceas al registrului de deplasare, care aduce bitul B_7 în poziția cea mai din stînga și elimină bitul de START pe la dreapta, numărătorul modulo 9 revine în starea **I** și din nou ENDR = „1"; aceasta face ca în starea sa **G**, următoare furnizării ultimului impuls de ceas, numărătorul modulo 7 să producă semnalul STB care transferă codul caracterului în circuitul 8212 și poziționează RXRDY. În continuare, el evoluează liber pînă în starea **C**, unde rămîne blocat. Remarcăm că logica de comandă nu testează recepționarea corectă a tuturor biților de STOP, ceea ce crește gradul de generalitate al schemei, ea putînd recepționa în condiții similare informația transmisă în format 8/11 cu doi biți de STOP, ca și informația transmisă în format 8/10 cu un singur bit de STOP.

Implementarea practică a registrului de deplasare este făcută cu circuite 7495, iar pentru numărătoarele de stare se poate folosi orice tip de numărător sincron.

Circuitele de interfață pentru adaptarea la cerințele standardului CCITT V24 (EIA RS-232 C) sau la sistemul de transmisie în buclă de curent de 20 mA sînt prezentate în figura 6.6.

Pentru exploatarea acestei interfețe sînt folosite trei subrutine: CI (Console Input), CO (Console Output) și TI (Tape Input).

După cum am mai amintit la § 4.3.2. aceste subrutine lucrează în buclă programată, conform principiului testării în buclă a biților de stare ai interfeței prezentat în organigramele din figura 4.28 și figura 4.29.

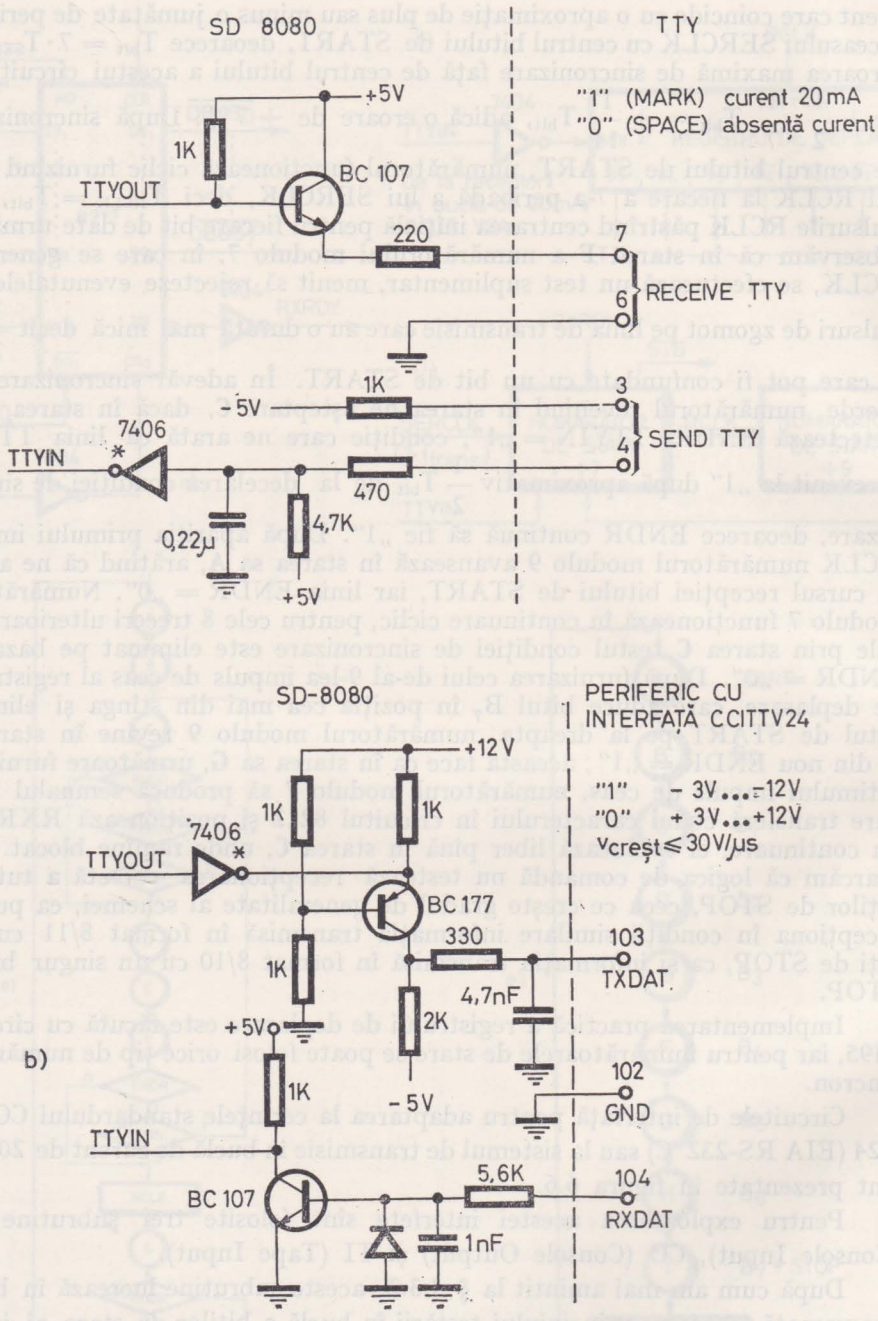


Fig. 6.6. Circuitele de interfață pentru transmisia în buclă de 20 mA sau niveluri CCITT V24

Dăm mai jos programul aferent rutinei CI.

```
;RUTINA CI RECEPȚIØNEAZĂ UN CARACTER DE LA
;CØNSØLĂ ȘI ÎL RETURNEAZĂ PRØGRAMULUI CHEMĂTØR
;ÎN ACUMULATØR
;DISTRUGE: A, INDICATØRII
CI:     IN      IØST    ;ADU ØCTETUL DE STARE I/E
        ANI     CIMSK   ;REȚINE BITUL RXRDY ȘI PØZIȚIØ-
                        ;NEAZĂ
        JZ      CI      ;INDICATØRII — AȘTEAPTĂ CARACTER
        IN      CNSI    ;ADU CARACTERUL ÎN ACUMULATØR
        RET             ;RETUR ÎN PRØGRAMUL CHEMĂTØR
IØST    EQU     0FDH    ;ADRESA ØCTETULUI DE STARE I/E
CIMSK   EQU     2       ;MASCA PT. BITUL 1=RXRDY
CNSI    EQU     0FEH    ;ADRESA DESERIALIZØRULUI
```

Programul aferent rutinei CØ este următorul:

```
;RUTINA CØ TRIMITE LA CØNSØLĂ CARACTERUL PREGĂTIT
;DE PRØGRAMUL CHEMĂTØR ÎN REGISTRUL C
;DISTRUGE: A, INDICATØRII
CØ:     IN      IØST    ;ADU ØCTETUL DE STARE I/E
        ANI     CØMSK   ;PØZIȚIØNEAZĂ INDICATØRUL Z =
                        ;= TXRDY
        JZ      CØ      ;AȘTEAPTĂ SERIALIZØR GATA
        MØV     A,C     ;MUTĂ CARACTERUL ÎN ACUMULATØR
        ØUT     CNSØ    ;TRANSFERĂ CARACTER LA SERIALIZØR
        RET             ;RETUR ÎN PRØGRAMUL CHEMĂTØR
IØST    EQU     0FDH    ;ADRESA ØCTETULUI DE STARE I/E
CØMSK   EQU     1       ;MASCA PT. BITUL 0=TXRDY
CNSØ    EQU     0FEH    ;ADRESA SERIALIZØRULUI
```

Rutina TI aduce în acumulator un caracter citit de pe banda perforată montată în cititorul consolei, în cazul respectiv un Teletype ASR 33 sau echivalent. Condiția de operare ce trebuie întrunită pentru funcționarea acestei rutine este ca cititorul de bandă să fie activat punîndu-se comutatorul lui de pornire în poziția START înainte de chemarea rutinei TI; în acest fel comanda de citire-caracter trimisă de rutina TI prin bitul 0 al adresei de ieșire FD_{16} găsește circuitul de acționare a magnetului selector al cititorului de bandă pregătit (închis), realizîndu-se avansul cu un pas prin anclanșarea temporară a releului comandat din calculator. Punerea comutatorului de pornire în poziția START după trimiterea comenzii de avans un pas prin rutina TI este inoperantă, deoarece această comandă are un caracter temporar, după epuizarea ei circuitul magnetului selector fiind potențial deschis.

```
;RUTINA TI ADUCE ÎN ACUMULATØR UN CARACTER CITIT
;DE PE BANDĂ PERFØRATĂ
;DISTRUGE: A, INDJCATØRII
```

```
TI:     MVI   A,TRC  ;CØMANDA DE CITIRE BANDĂ
        ØUT   IØCTR
        CALL  CI     ;ADU CARACTERUL IN ACUMULATØR
        RET          ;REVENIRE
IØCTR:  EQU   0FDH   ;ADRESA ØCTET CØMANDĂ I/E
TRC     EQU   1      ;CDA.CIT. BANDĂ—BIT 0=1
```

În cazul unei console care posedă cititor/perforator de bandă se poate executa și operația de generare de benzi perforate, folosindu-se pentru aceasta rutina CØ care tipărește și perforează simultan.

Vom da acum un exemplu practic de folosire a acestor rutine. Este vorba de rutina CHAR care efectuează următoarele funcțiuni:

1. Aduce în acumulator caracterul tastat de operator de la consolă; acest caracter nu este tipărit sau afișat pe consolă în mod local, transmisia efectuîndu-se *full duplex*.

2. Considerînd că lucrăm în cod ASCII cu paritate pară se efectuează verificarea de paritate a caracterului, în sensul că trebuie să avem:

$$B_7 \oplus B_6 \oplus B_5 \oplus B_4 \oplus B_3 \oplus B_2 \oplus B_1 \oplus B_0 = 0$$

3. În caz de recepție corectă se reține numai biții codului caracterului B_0—B_6, făcîndu-se $B_7 = 0$. Caracterul reținut e trimis înapoi la consolă pentru tipărire sau afișare, în ecou. Se testează, de asemenea, dacă acest caracter nu este cumva caracterul special ESC (Escape), cod $1B_{16}$, în care caz se trimite în ecou caracterul „$", cod 24_{16}, poziționîndu-se la ieșire indicatorul CARRY=„1".

4. În caz de recepție incorectă se trimite în ecou caracterul „@" pentru a indica operatorului că a fost o eroare de paritate și se așteaptă primirea unui caracter corect.

```
; RUTINA CHAR DE RECEPȚIE CARACTERE ASCII DE LA CØNSØLĂ
;    —FACE VERIFICAREA DE PARITATE PARĂ
;    —MASCHEAZĂ BITUL DE PARITATE B7
;    —TIPĂREȘTE LA CØNSØLĂ ÎN ECØU
;    —PØZIȚIØNEAZĂ CARRY=1 LA RECEPȚIØNAREA ESC
; DISTRUGE: A,B,C, INDICATØRII
CHAR:   CALL  CI     ;ADU CARACTER TASTAT ÎN A
        ØRA   A      ;PØZIȚIØNEAZĂ INDICATØR PARITATE
        JPE   CHAR1  ;TESTEAZĂ PARITATEA
        MVI   C,MKY  ;RECEPȚIE INCØRECTĂ, @ ÎN ECØU
        CALL  CØ
        JMP   CHAR   ;AȘTEAPTĂ CARACTER CØRECT
CHAR1:  MVI   B,7FH  ;RECEPȚIE CØRECTĂ, B7=0
        ANA   B
        MVI   B,ESC  ;TEST DACĂ ESCAPE
        CMP   B
        JZ    CHAR2
        MØV   C,A    ;CARACTER NØRMAL, ECØU
        CALL  CØ
```

```
        ØRA   A          ;CARRY=0
        RET              ;RETUR NØRMAL
CHAR2:  MVI   C,DØL      ;$ ÎN ECØU
        CALL  CØ
        STC              ;CARRY=1
        RET              ;RETUR PE ESCAPE.
ESC     EQU   1BH        ;CØDUL CARACTERULUI ESC
DØL     EQU   24H        ;CØDUL CARACTERULUI $
MKY     EQU   40H        ;CØDUL CARACTERULUI @
```

În încheierea paragrafului va trebui să remarcăm că varianta de implementare cea mai economică a acestei interfețe se obține folosind circuitul LSI specializat, Intel 8251 (Universal Synchronous/Asynchronous Receiver/Transmitter), care, așa cum vom vedea în paragraful destinat acestui circuit, permite programarea numărului biților de STOP, ca și tipul controlului de paritate. Am considerat însă indicată prezentarea detaliată a acestei interfețe-serie realizată cu circuite mai simple, pentru familiarizarea mai profundă a cititorului cu problematica proiectării hardware a unei interfețe de I/E.

6.2. INTERFAȚA PENTRU PERFORATORUL DE BANDĂ

Aceasta este o interfață de ieșire paralelă pe 8 biți, din cadrul SD-8080, care folosește adresa de ieșire FB_{16} pentru octetul de date și bitul 2=PCHRDY (PUNCH READY) al octetului de stare I/E recepționat de procesor pe adresa de intrare FD_{16} (vezi tabelele 4.3 și 4.4).

În proiectarea acestei interfețe se pornește de la specificațiile de interfață ale perforatorului de bandă FACIT 4070 folosit; el comunică cu exteriorul prin următoarele semnale:

— CH1-CH9 sînt semnalele de comandă a poansoanelor, CH1 corespunzînd bitului cel mai puțin semnificativ al octetului de date, iar CH8 bitului cel mai semnificativ (fig. 4.29*c*); CH9 corespunde perforației de sincronizare care în cazul nostru va fi totdeauna folosită.

— PI (PUNCH INSTRUCTION) este semnalul care comandă logica perforatorului pentru acționarea poansoanelor indicată prin conținutul octetului de pe liniile CH1-CH8.

— PR (PUNCH READY) este semnalul care indică sistemului faptul că perforatorul întrunește toate condițiile necesare pentru a primi date în vederea perforării.

Semnalele CH1-CH9 și PI sînt furnizate de către sistem perforatorului de bandă și sînt supuse următoarelor reguli din punct de vedere electric:

— „1" nivel cuprins între +3,5V și +12V;
— „0" nivel cuprins între —12V și +1,5V;
— impedanță de intrare 22 Kohmi.

Semnalul PR este furnizat de perforator sistemului și are următoarele niveluri electrice:

— „1" nivel +6V, impedanță de ieșire 1 Kohm;

— „0" max. 0,4 V, max. 10 mA.

Cronograma semnalelor de interfață ale perforatorului e prezentată în figura 6.7. Duratele impulsurilor corespund valorilor realizate în cazul implementării reale, în timp ce valorile între paranteze, date pentru PI și EDTA sînt limitele impuse de fabricantul dispozitivului. Observăm că intervalul de timp scurs de la trimiterea unei comenzi de perforare pînă la reîntoarcerea perforatorului în starea „pregătit", perioada lui PR de pe cronogramă, este de 13,3 ms, corespunzînd ratei maxime de transfer de 75 car/s a dispozitivului.

Conectarea interfeței la magistrala de date se face folosind un circuit 8212 configurat în modul de ieșire similar cu circuitul prezentat în figura 6.4*a*. Intrarea $\overline{DS_1}$ e în acest caz legată la $ABUS_2$ pentru a se realiza selectarea circuitului în timpul executării instrucțiunii ØUT 0FBH pe principiul „selecției liniare"; liniile de ieșire de date DO_1—DO_8 se numesc în acest caz $PCHD_0$—$PCHD_7$, în timp ce linia $\overline{INT}$ pe care apare un impuls negativ de 500 ns în timpul selecției, poartă numele $\overline{GOPCH}$ și atacă pe intrarea de forțare la „1" bistabilul de acționare a interfeței AUTH (fig. 6.8*a*). Semnalul AUTH validează ceasul PCL1MHZ cu frecvența de 1 MHz obținut prin divizarea cu 2 a ceasului de 2MHz, Ø1, al sistemului, iar conjuncția lui $\overline{AUTH}$ cu PRDY,

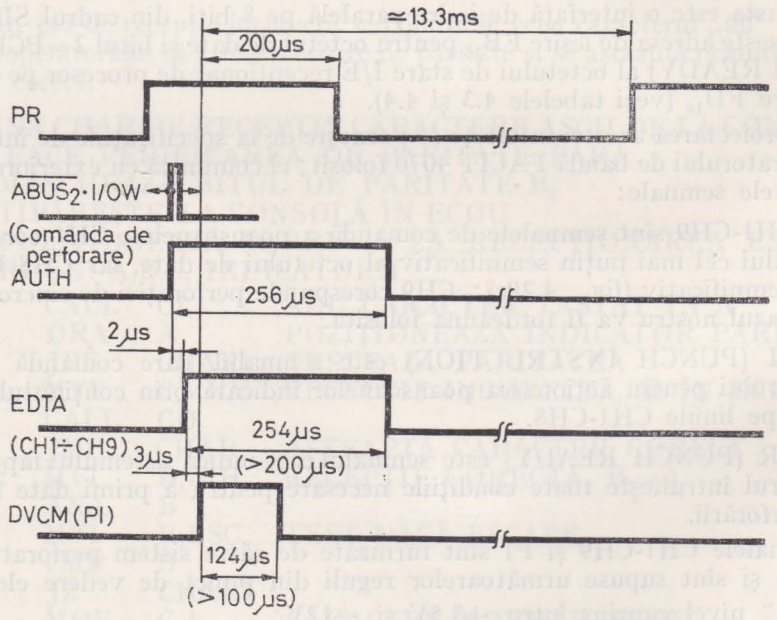


Fig. 6.7. Cronograma semnalelor de interfață ale perforatorului de bandă

repetarea lui PR primit de la perforator, formează semnalul de stare al interfeței PCHRDY trimis pe bitul 2 al octetului de stare a I/E prin circuitul 8226 folosit pentru TXRDY și RXRDY (fig. 6.3).

Secvențierea în timp a activității interfeței după activarea ei prin instrucțiunea ØUT 0FBH ce poziționează AUTH este realizată cu ajutorul unui numărător de stări modulo 256, funcționînd pe baza ceasului PCL1MHz, ce furnizează semnalele DVCM (echivalent cu PI) și EDTA de validare a liniilor $PCHD_i$ pe liniile CH_{i+1}, i = 0—7. Numărătorul de stări e prezentat în figura 6.8*b*, iar formarea semnalelor EDTA și DVCM în figura 6.8*c*. Pe schema din figura 6.8*d* este prezentată formarea semnalului RAUTH care identifică starea **255** a numărătorului permițînd dezactivarea lui AUTH, și deci a întregii interfețe, după 256 μs de la pornire. În figura 6.8*d* e prezentată, de asemenea, generarea semnalelor de cablu CH1÷CH9 ale perforatorului pe baza validării liniilor $PCHD_0$—$PCHD_7$ cu ajutorul strobului EDTA (Enable Data).

Revenind la cronograma din figura 6.7 observăm că pornirea activității interfeței pentru perforarea unui caracter se face prin poziționarea lui AUTH pe baza condiției de selecție a circuitului 8212 al interfeței $\overline{ABUS_2} \cdot I/OW$ apărută în timpul executării instrucțiunii ØUT 0FBH. În acest moment este autorizat ceasul PCL1MHz care acționează numărătorul de stări PST_0-PST_7. În starea 2 a numărătorului se poziționează semnalul EDTA care activează liniile de date ale perifericului. Întîrzierea de 2 μs între AUTH și EDTA este prevăzută pentru a fi siguri de corectitudinea liniilor $PCHD_i$ în momentul transmiterii lor către dispozitiv.

În continuare, în starea **5** a numărătorului apare semnalul DVCM (Device Command) care se transmite la periferic ca PI, întîrzierea de 3 μs de la date la semnalul PI fiind prevăzută ca interval de acoperire a datelor. În starea **129** PI dispare, asigurîndu-se astfel o durată de 124 μs, intervalul minim de timp pentru care trebuie să existe acesta fiind de 100 μs. Semnalele EDTA și AUTH dispar după starea **255** a numărătorului, pe baza lui RAUTH. Activitatea interfeței se termină în acest moment deoarece dispariția lui AUTH invalidează ceasul PCL1MHz, iar numărătorul de stări rămîne blocat în starea **0** prin absența ceasului. Referitor la comportarea lui PCHRDY observăm că acesta este „1", semnalînd că sînt reunite toate condițiile pentru perforare, dacă PR = „1" și AUTH = „0", adică în momentul în care perifericul și interfața sînt ambele pregătite. Testînd acest semnal, procesorul trimite instrucțiunea ØUT 0FDH care transferă caracterul de date interfeței și pornește activitatea ei prin AUTH=„1". Din acest moment PCHRDY=„0", semnalul fiind condiționat de $\overline{AUTH}$. El e menținut la „0" de $\overline{AUTH}$=„0" pentru cele 256 μs cît interfața își execută ciclul de funcționare, dar continuă să rămînă „0" și după ce AUTH revine la „0" ($\overline{AUTH}$=„1") pe baza condiției PR=„0" care are loc la aproximativ 200 μs de la trimiterea lui PI către periferic. La eliberarea perifericului după scurgerea celor 13,3 ms afectate perforării unui caracter, PR redevine „1" autorizînd PCHRDY=„1".

În ceea ce privește programarea punerii în funcțiune a acestui periferic se folosește principiul „buclei programate" despre care s-a mai vorbit.

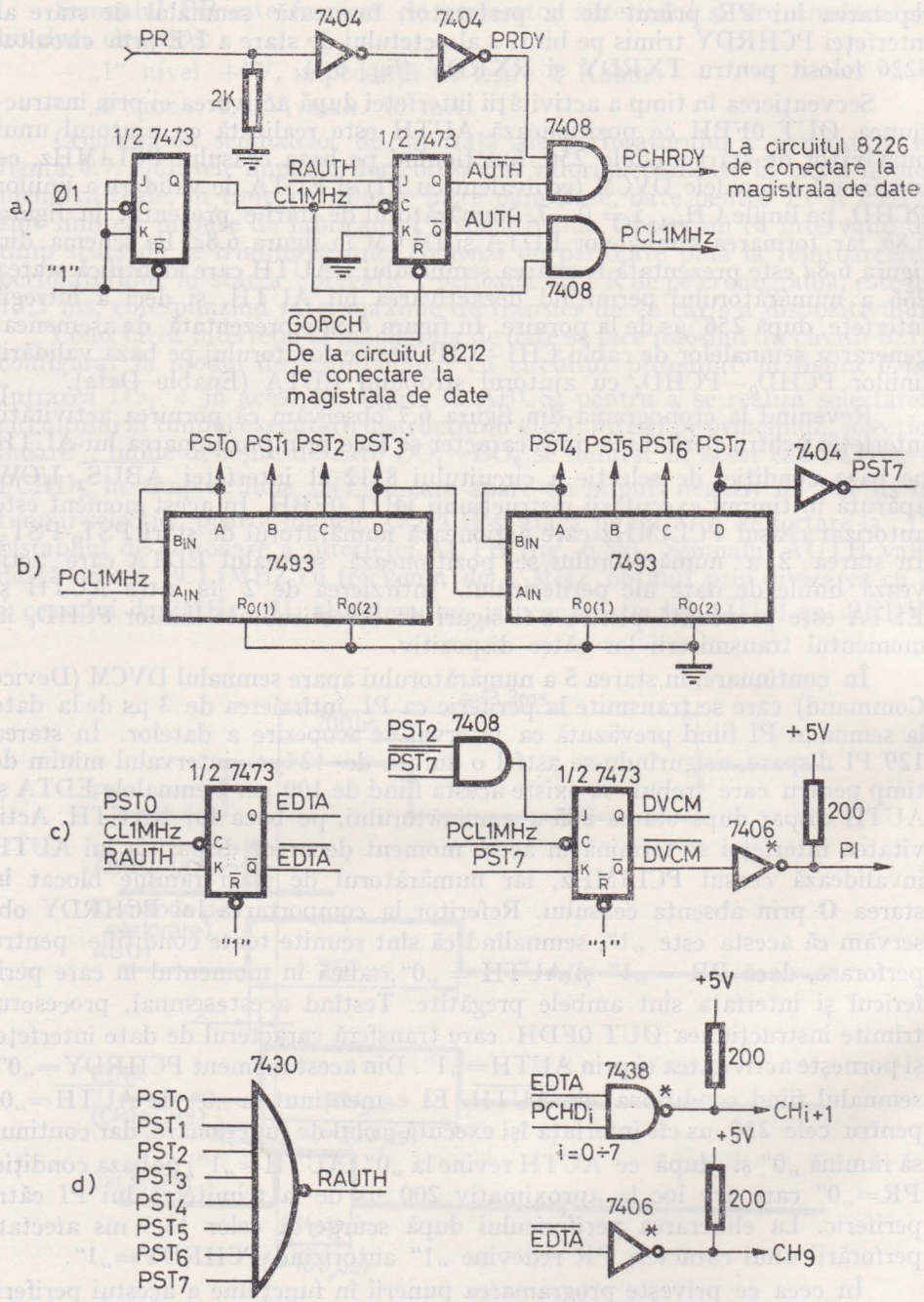


Fig. 6.8. Schemele interfeței pentru perforatorul de bandă

Subrutina PØ (PUNCH ØUTPUT) aferentă acestui periferic respectă structura descrisă în organigrama din figura 4.28. Ea primește codul caracterului de perforat în registrul C, așteaptă pînă cînd perifericul și interfața sînt gata de lucru, transferă codul caracterului către interfață și revine în programul principal. Secvența de program aferentă subrutinei PØ e dată mai jos.

```
; SUBRUTINA DE PERFØRARE CARACTER PE BANDĂ.
; PRIMEȘTE CØDUL DE PERFØRAT ÎN REGISTRUL C.
; DISTRUGE: A, INDICATØRII
PØ:     IN      IØST    ;ADU ØCTETUL DE STARE AL I/E
        ANI     PCMSK   ;REȚINE BITUL PCHRDY
        JZ      PØ      ;AȘTEAPTĂ PERIFERIC GATA
        MØV     A,C     ;TRANSFERĂ CARACTER
        ØUT     PCHØ
        RET             ;REVENIRE
IØST    EQU     0FDH    ;ADRESA ØCTET STARE I/E
PCMSK   EQU     4       ;MASCA PT. BITUL 2=PCHRDY
PCHØ    EQU     0FBH    ;ADRESA PERFØRATØR BANDĂ
```

6.3. INTERFAȚA PENTRU CITITORUL DE BANDĂ PERFORATĂ

Aceasta este o interfață de intrare paralelă pe 8 biți din cadrul SD-8080 care folosește adresa de intrare FB_{16} pentru octetul de date și bitul 3 RDRDY (READER READY) al octetului de stare primit pe adresa de intrare FD_{16} (tabelele 4.3 și 4.4).

Semnalele de interfață ale cititorului FACIT 4021 folosit sînt următoarele:

— D1-D8 sînt liniile de date provenind de la stația de citire fotoelectrică a dispozitivului, unde D1 corespunde bitului cel mai puțin semnificativ al codului (fig. 4.29*c*).

— RR (Reader Ready) este semnalul prin care cititorul indică faptul că are un caracter în avans citit de pe bandă, pregătit în buffer-ul său de ieșire și că acesta poate fi furnizat la cererea procesorului; D1-D8 și RR au următoarele specificații electrice: „1" corespunde unui nivel de +5V transmis printr-o impedanță de 2,2 Kohmi în timp ce „0" corespunde unui nivel de maximum 0,4 V, curentul injectat nedepășind 10 mA.

— RI (Read Instruction) este semnalul de cerere de către sistem a unui caracter de la cititorul de bandă; pentru acesta „1" este constituit de un nivel cuprins între +3,5V și +12V, iar „0" de un nivel cuprins între —12V și +1,5V, impedanța de intrare fiind de minimum 22 Kohmi.

— DA (Data Available) este strobul datelor prin care cititorul semnalează sistemului validitatea liniilor D1 la D8; fiind un semnal generat de cititor, el are aceleași specificații electrice ca și D1-D8 și RR.

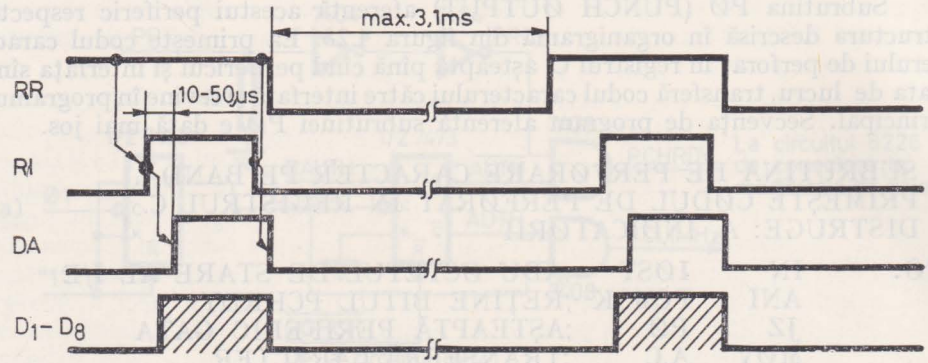


Fig. 6.9. Cronograma semnalelor de interfață ale cititorului de bandă perforată

Cronograma funcționării acestor semnale de interfață este prezentată în figura 6.9. Intervalul de timp maxim scurs de la căderea lui RR pînă la repunerea lui după pregătirea unui nou caracter este de 3,1 ms, corespunzînd unei viteze maxime de citire de aproximativ 300 car/s.

Conectarea interfeței la magistrala de date se face cu un circuit 8212 configurat în modul de intrare (MD=„0") similar cu circuitul prezentat în figura 6.5*a*. Selectarea acestui circuit se realizează prin condiția $\overline{ABUS_2} \cdot I/OR$ în timpul executării instrucțiunii IN 0FBH. Memorarea în 8212 a liniilor de date $\overline{D1}$-$\overline{D8}$, negarea liniilor D1-D8 recepționate în niveluri TTL de la periferic, se face pe frontul negativ al semnalului STROB, generat de logica de comandă a interfeței, conectat la intrarea STB a circuitului. Ieșirea sa $\overline{INT}$ constituie semnalul de stare a interfeței $\overline{RDRDY}$ care după negare se transmite pe bitul 3 al magistralei de date cu ajutorul circuitului 8226 prezentat în figura 6.3.

Logica de comandă a interfeței care furnizează semnalele RI și STROB este un automat secvențial funcționînd conform organigramei prezentate în figura 6.10*a*. Automatul, implementat cu ajutorul bistabililor RD2 și RD1, primește intrările asincrone RR și DA, în timp ce intrarea RDRDY este considerată sincronă în raport cu ceasul Ø1 al automatului, ea intervenind totdeauna la un moment de timp bine determinat față de acest ceas. El generează RI în starea **B**, după ce în starea **A** s-a asigurat că cititorul are un caracter pregătit, rămînînd în **B** pînă cînd cititorul semnalează prin DA prezența datelor valide pe liniile D_i ($i = 1-8$). În starea **C** următoare se introduc datele în circuitul 8212 cu ajutorul semnalului STROB, după care în **D** automatul așteaptă preluarea datelor înscrise în 8212 de către procesor pentru a-și putea relua ciclul de funcționare. Observăm că tranzițiile care depind de condiții de intrare asincrone au loc între stări cărora li s-au atribuit coduri adiacente, diferind printr-o singură variabilă de stare, ceea ce are rolul de a preveni funcționarea automatului în condiții de cursă.

Schema care implementează această organigramă este prezentată în figura 6.10*b*. Remarcăm că pentru recepția semnalului RR de pe cablu nu

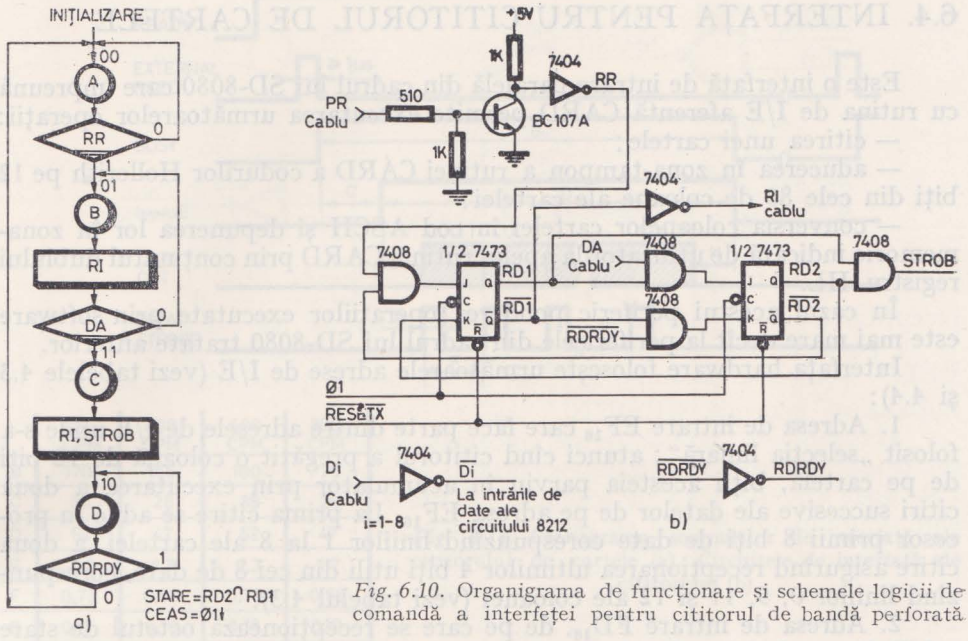


Fig. 6.10. Organigrama de funcționare și schemele logicii de comandă a interfeței pentru cititorul de bandă perforată

s-a folosit o simplă poartă TTL, ca în cazul lui DA sau D1-D8, deoarece în situația perifericului deconectat sau al cablului demontat s-ar primi RR=„1".

Programarea acestei interfețe se bazează pe testarea în buclă a bitului de stare al interfeței RDRDY, care se transmite pe bitul 3 al octetului de stare a I/E recepționat de sistem pe adresa de intrare FD_{16}, urmată de preluarea în acumulator a codului caracterului citit de pe bandă, prin adresarea circuitului de date al interfeței aflat pe adresa FB_{16}.

Rutina RI (Reader Input), a cărei structură coincide celei descrise în organigrama din figura 4.29, efectuează această operație predînd programului chemător în acumulatorul A codul caracterului citit de pe bandă. Secvența de instrucțiuni ce compun această rutină este dată mai jos.

```
;SUBRUTINA DE CITIRE CARACTER PE CITITØRUL
;DE BANDĂ — CØDUL CITIT E MEMØRAT ÎN A LA IEȘIRE
;DISTRUGE: A, INDICATØRII
RI:     IN      IØST    ;ADU ØCTETUL DE STARE A I/E
        ANI     RIMSK   ;REȚINE BITUL RDRDY
        JZ      RI      ;AȘTEAPTĂ CITITØR GATA
        IN      RDRI    ;DU CARACTER DE DATE
        RET             ;AREVENIRE
IØST    EQU     0FDH    ;ADRESA ØCTET DE STARE I/E
RIMSK   EQU     8       ;MASCA PENTRU BITUL 3=RDRDY
RDRI    EQU     0FBH    ;ADRESA CITITØR BANDĂ
```

6.4. INTERFAȚA PENTRU CITITORUL DE CARTELE

Este o interfață de intrare paralelă din cadrul lui SD-8080 care împreună cu rutina de I/E aferentă CARD permite executarea următoarelor operații:

— citirea unei cartele;

— aducerea în zona-tampon a rutinei CARD a codurilor Hollerith pe 12 biți din cele 80 de coloane ale cartelei;

— conversia coloanelor cartelei în cod ASCII și depunerea lor în zona-memorie indicată de utilizator la apelul rutinei CARD prin conținutul dublului registru HL.

În cazul acestui periferic ponderea operațiilor executate prin software este mai mare decît la perifericele din cadrul lui SD-8080 tratate anterior.

Interfața hardware folosește următoarele adrese de I/E (vezi tabelele 4.3 și 4.4):

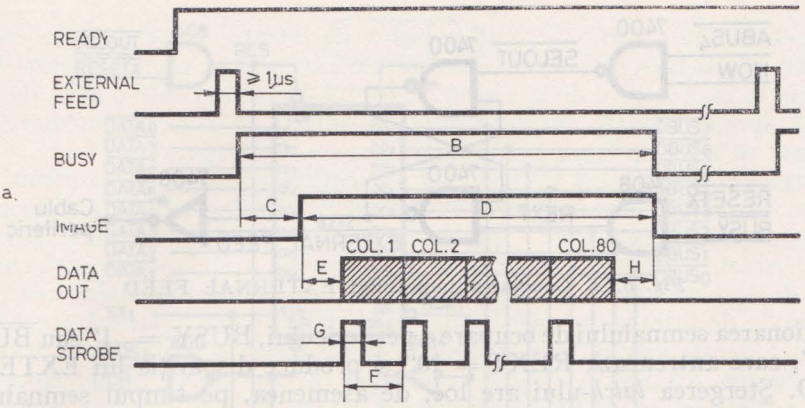
1. Adresa de intrare EF_{16} care face parte dintre adresele de I/E unde s-a folosit „selecția liniară"; atunci cînd cititorul a pregătit o coloană de 12 biți de pe cartelă, biții acesteia parvin în acumulator prin executarea a două citiri succesive ale datelor de pe adresa EF_{16}. La prima citire se aduc în procesor primii 8 biți de date corespunzînd liniilor 1 la 8 ale cartelei, a doua citire asigurînd recepționarea ultimilor 4 biți utili din cei 8 de date, corespunzînd liniilor 0, 9, 11 și 12 ale coloanei (vezi tabelul 4.3).

2. Adresa de intrare FD_{16}, de pe care se recepționează octetul de stare a I/E, în cazul cititorului de cartele interesîndu-ne biții: 4 = CRRDY (CARD READER READY), semnificînd atunci cînd este „1" că cititorul este pregătit de lucru din toate punctele de vedere (pus sub tensiune, are cartele în magazia de intrare, este inițializat etc.); 5 = DATRDY (DATA STROBE) are semnificația unui semnal de validare a liniilor de date; 6 = CRBUSY (CARD READER BUSY) indică, atunci cînd este „1", că perifericul se află în curs de citire a unei cartele.

3. Adresa de ieșire EF_{16}; executarea unei instrucțiuni ØUT 0EFH are ca efect trimiterea impulsului de comandă a cititorului, EXTERNAL FEED, producînd antrenarea unei cartele și trecerea ei prin fața stației de citire.

Funcționarea cititorului din punctul de vedere al semnalelor sale de interfață cu sistemul de dezvoltare este prezentată în diagrama de impulsuri din figura 6.11*a*.

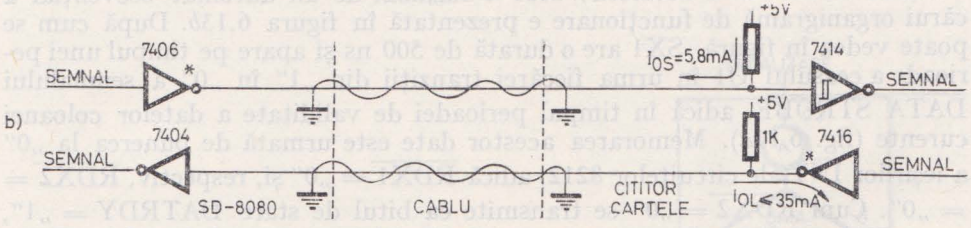
La apariția semnalului de stare READY al cititorului, semnificînd că el întrunește toate condițiile de lucru, transmis ca bitul de stare CRRDY pe octetul de stare a I/E cu ajutorul unui circuit 8226 similar cu cel prezentat în figura 6.3, sistemul de dezvoltare trimite impulsul EXTERNAL FEED care inițializează ciclul de citire a unei cartele. Ocuparea cititorului este indicată cu ajutorul semnalului BUSY; EXTERNAL FEED dispare ca urmare a apariției lui BUSY. În timpul citirii cartelei apare de 80 de ori impulsul DATA STROBE validînd datele de pe liniile DATA 0—9, DATA 11 și DATA 12 corespunzînd coloanelor cartelei. Eliberarea cititorului după terminarea citirii cartelei este semnalată prin revenirea lui BUSY la „0" lucru de care procesorul ia cunoștință testînd bitul de stare CRBUSY. Observăm că dacă nu se produce nici un incident în cursul citirii cartelei, semnalul READY este



	300 CPM	400 CPM	600 CPM	800 CPM
B	170	165	100	75
C	38	28	38	28
D	62	47	62	47
E	2,3	1,7	2,3	1,7
F	0,73	0,56	0,73	0,56
G	0,13	0,10	0,13	0,10
H	2,0	1,5	2,0	1,5

TIMPII IN MS

Fig. 6.11. Cronograma semnalelor de interfață ale cititorului de cartele (a). Circuitele de interfață ale cititorului (b)



permanent menținut la „1"; el trece imediat la „0" în cazul oricărui incident: suprapunerea mai multor cartele în stația de citire, dubla încercare de alimentare a unei cartele rămasă fără efect, magazia de intrare goală, magazia de ieșire plină, eroare de sincronizare pe coloane în timpul citirii.

Din punct de vedere electric, semnalele de interfață se transmit în niveluri TTL obișnuite, circuitele de comandă și recepție ale cititorului și sistemului de dezvoltare fiind prezentate în figura 6.11*b*.

Schema de generare a impulsului EXTERNAL FEED e prezentată în figura 6.12. Acest semnal este produs de un *latch* poziționat de $\overline{\text{SELOUT}}$ = „0" unde:

$$\overline{\text{SELOUT}} = \overline{\text{ABUS}_4 \cdot \text{I/OW}},$$

este un impuls negativ cu durata de 500 ns, ce apare în cursul stării T_3 a ciclului de IEȘIRE al instrucțiunii ØUT 0EFH. *Latch*-ul rămîne poziționat pînă la

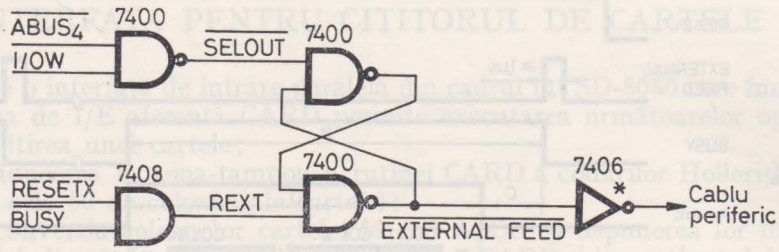


Fig. 6.12. Generarea impulsului EXTERNAL FEED

recepționarea semnalului de ocupare a perifericului, BUSY = „1" sau $\overline{BUSY}$ = = „0", care antrenează REXT = „0" și produce dispariția lui EXTERNAL FEED. Ștergerea *latch*-ului are loc, de asemenea, pe timpul semnalului de inițializare generală $\overline{RESETX}$.

Transmiterea datelor de pe coloanele cartelei la procesor sub forma a doi octeți succesivi e realizată cu ajutorul schemei prezentate în figura 6.13*a*. Cele două circuite 8212 ale schemei, conectate în mod de intrare (MD = „0"), sînt simultan inițializate cu ajutorul semnalului RES, activ jos, care apare fie în timpul inițializării generale a sistemului (componenta $\overline{RESETX}$), fie la trimiterea de la procesor a comenzii de citire a unei cartele (componenta $\overline{SELOUT}$). Aceste circuite joacă și rolul de receptori de cablu pentru semnalele $\overline{DATA_i}$ ale cititorului de cartele. Memorarea conținutului coloanei curente în circuitele 8212 are loc pe frontul negativ al semnalului SX1 conectat pe intrarea STROBE a acestora; SX1 e furnizat de un automat secvențial a cărui organigramă de funcționare e prezentată în figura 6.13*b*. După cum se poate vedea în figură, SX1 are o durată de 500 ns și apare pe timpul unei perioade a ceasului Ø1 în urma fiecărei tranziții din „1" în „0" a semnalului $\overline{DATA\ STROBE}$, adică în timpul perioadei de validitate a datelor coloanei curente (fig. 6.11*a*). Memorarea acestor date este urmată de punerea la „0" a ieșirilor $\overline{INT}$ ale circuitelor 8212, adică $\overline{RDX1}$ = „0" și, respectiv, $\overline{RDX2}$ = = „0". Cum $\overline{RDX2}$ = „0" se transmite ca bitul de stare DATRDY = „1", procesorul va efectua în acest moment două citiri succesive de la adresa EF_{16}. La prima citire este selectat primul circuit 8212 deoarece primește „1" pe intrarea de selecție DS_2; ca urmare bistabilul intern „Service Request" se șterge, ieșirea sa $\overline{INT}$ ce poartă numele $\overline{RDX1}$ devine „1", permițînd selectarea celui de-al doilea circuit 8212 în timpul celei de-a doua citiri inițiate de procesor. După aceasta se șterge și bistabilul „Service Request" al celui de-al doilea circuit, astfel că în continuare DATRDY = „0". Constatăm că ieșirile de date DO ale celor două circuite 8212 sînt conectate în paralel pe magistrala de date $\overline{DBUS_0}$ — $\overline{DBUS_7}$.

Interfața hardware este activată cu ajutorul *driver*-ului cititorului de cartele, rutina CARD, care execută funcțiunile amintite la începutul paragrafului. Rutina CARD folosește două buffere: unul în care depune cele 80 coloane ale cartelei după conversia în cod ASCII și care e indicat de utilizator prin adresa încărcată de acesta în HL înainte de apelul rutinei, al doilea, cu

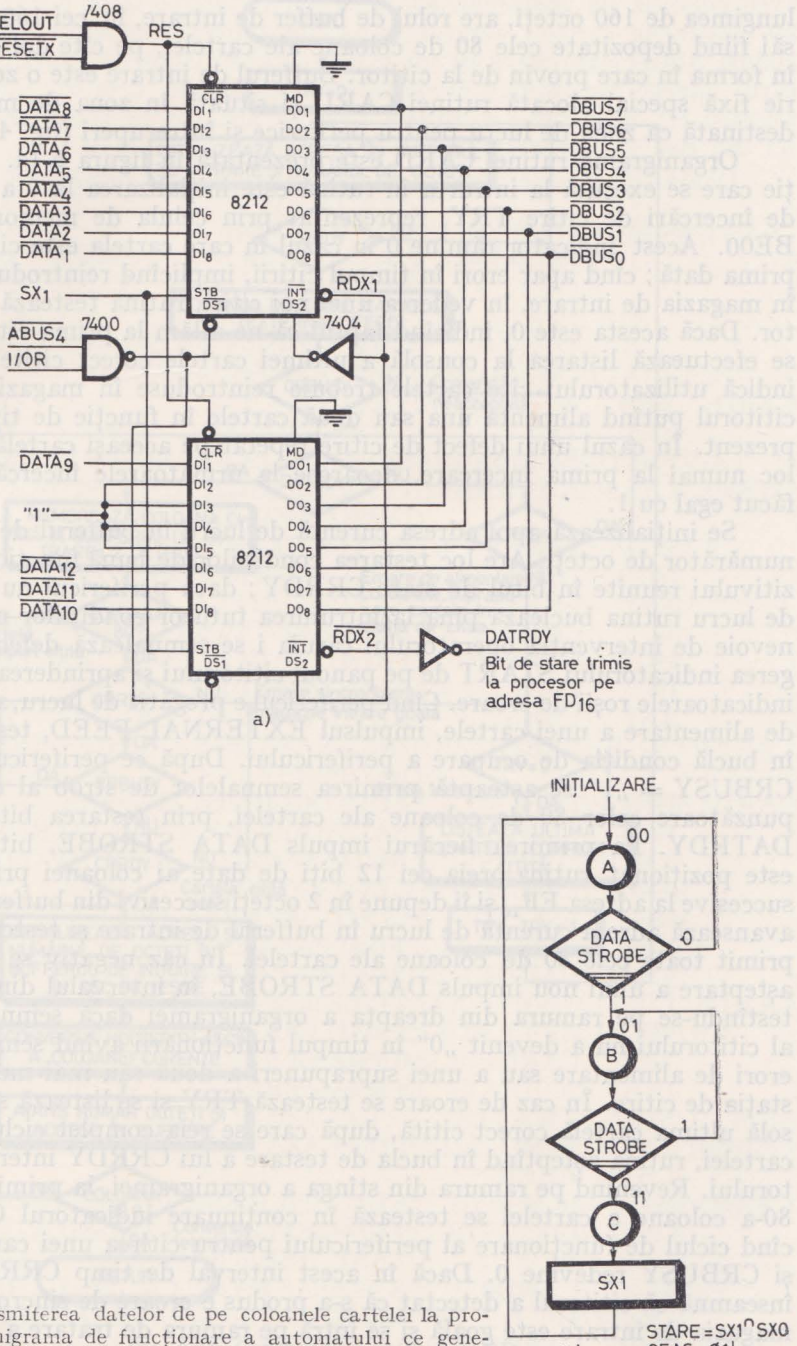


Fig. 6.13. Transmiterea datelor de pe coloanele cartelei la procesor (a). Organigrama de funcționare a automatului ce generează SX1 (b)

lungimea de 160 octeți, are rolul de buffer de intrare, în cei 160 de coteți ai săi fiind depozitate cele 80 de coloane ale cartelei, pe cîte doi octeți fiecare, în forma în care provin de la cititor. Bufferul de intrare este o zonă de memorie fixă special alocată rutinei CARD și situată în zona de memorie RAM destinată ca zonă de lucru pentru periferice și întreruperi (fig. 4.11).

Organigrama rutinei CARD este prezentată în figura 6.14. Prima operație care se execută la intrarea în rutină este inițializarea la 0 a indicatorului de încercări de citire TRY, reprezentat prin celula de memorie cu adresa BE00. Acest indicator rămîne 0 în cazul în care cartela este citită corect de prima dată; cînd apar erori în timpul citirii, implicînd reintroducerea cartelei în magazia de intrare, în vederea unei noi citiri, rutina testează acest indicator. Dacă acesta este 0, indicînd faptul că ne aflăm la prima citire a cartelei, se efectuează listarea la consolă a ultimei cartele corect citite. Acest lucru indică utilizatorului cîte cartele trebuie reintroduse în magazia de intrare, cititorul putînd alimenta una sau două cartele în funcție de tipul de eroare prezent. În cazul unui defect de citire repetat pe aceeași cartelă, listarea are loc numai la prima încercare, deoarece la următoarele încercări TRY este făcut egal cu 1.

Se inițializează apoi adresa curentă de lucru în bufferul de intrare și un numărător de octeți. Are loc testarea condițiilor de bună funcționare a dispozitivului reunite în bitul de stare CRRDY; dacă perifericul nu este pregătit de lucru rutina buclează pînă la întrunirea tuturor condițiilor necesare, fiind nevoie de intervenția operatorului căruia i se semnalează defectul prin stingerea indicatorului START de pe panoul cititorului și aprinderea unuia dintre indicatoarele roșii de eroare. Cînd perifericul e pregătit de lucru, se dă comanda de alimentare a unei cartele, impulsul EXTERNAL FEED, testîndu-se apoi în buclă condiția de ocupare a perifericului. După ce perifericul semnalează CRBUSY = „1" se așteaptă primirea semnalelor de strob al datelor corespunzătoare celor 80 de coloane ale cartelei, prin testarea bitului de stare DATRDY. La primirea fiecărui impuls DATA STROBE, bitul DATRDY este poziționat, rutina preia cei 12 biți de date ai coloanei prin două citiri succesive la adresa EF_{16} și îi depune în 2 octeți succesivi din bufferul de intrare, avansează adresa curentă de lucru în bufferul de intrare și testează dacă s-au primit toate cele 80 de coloane ale cartelei. În caz negativ se reia bucla de așteptare a unui nou impuls DATA STROBE, în intervalul dintre impulsuri testîndu-se pe ramura din dreapta a organigramei dacă semnalul READY al cititorului nu a devenit „0" în timpul funcționării avînd semnificația unei erori de alimentare sau a unei suprapuneri a două sau mai multe cartele în stația de citire. În caz de eroare se testează TRY și se listează sau nu la consolă ultima cartelă corect citită, după care se reia complet ciclul de citire al cartelei, rutina așteptînd în bucla de testare a lui CRRDY intervenția operatorului. Revenind pe ramura din stînga a organigramei, la primirea celei de-a 80-a coloane a cartelei se testează în continuare indicatorul CRRDY pînă cînd ciclul de funcționare al perifericului pentru citirea unei cartele ia sfîrșit și CRBUSY redevine 0. Dacă în acest interval de timp CRRDY devine 0 înseamnă că cititorul a detectat că s-a produs e eroare de sincronizare sau că magazia de intrare este goală și se intră pe ramura de tratare a erorii pe care am comentat-o anterior. Cînd ciclul de citire este epuizat fără a se fi detectat

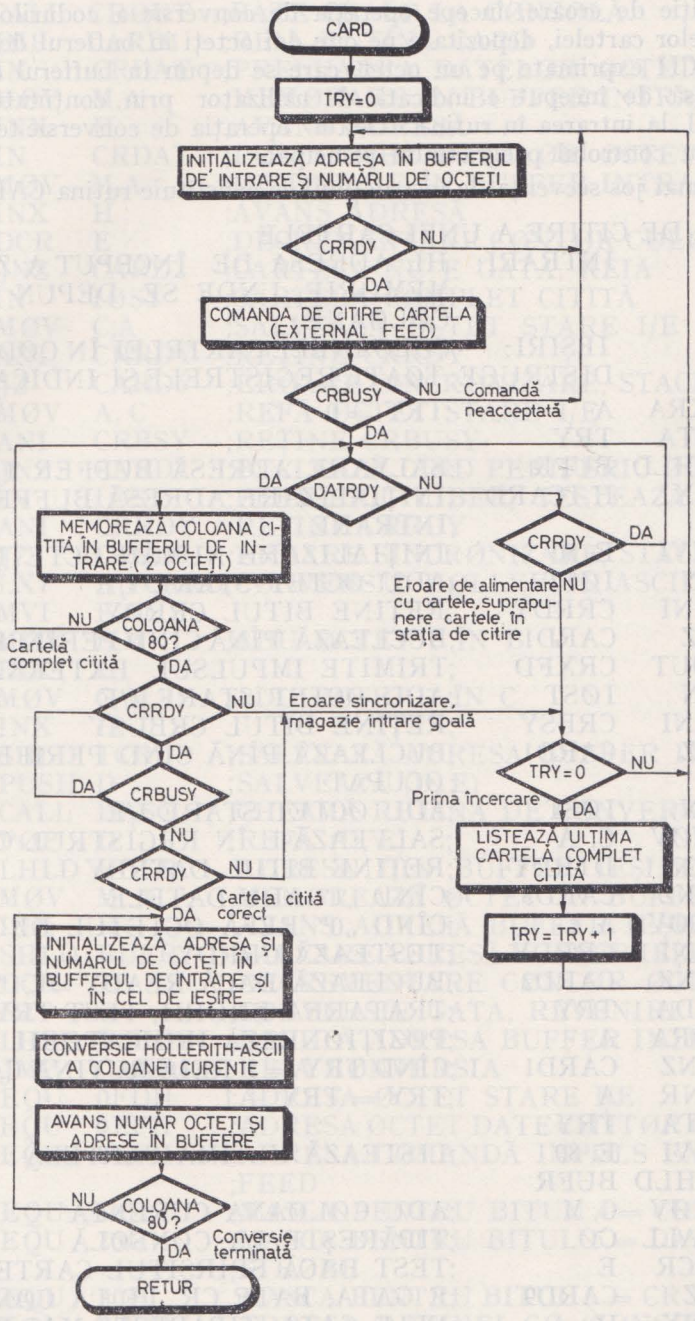


Fig. 6.14. Organigrama rutinei CARD

vreo condiție de eroare, începe operația de conversie a codurilor Hollerith ale coloanelor cartelei, depozitate pe cîte doi octeți în bufferul de intrare, în coduri ASCII exprimate pe un octet, care se depun în bufferul de ieșire, a cărui adresă de început e indicată de utilizator prin conținutul dublului registru HL la intrarea în rutină. Odată operația de conversie terminată se dă din nou controlul programului principal.

Dăm mai jos secvența de instrucțiuni ce constituie rutina CARD:

```
;RUTINA DE CITIRE A UNEI CARTELE
;                INTRĂRI:  HL ADRESA DE ÎNCEPUT A ZØNEI DE
;                          MEMØRIE UNDE SE DEPUN CELE 80
;                          CØLØANE
;                IEȘIRI:   CØLØANELE CARTELEI ÎN CØD ASCII
;                DISTRUGE: TØATE REGISTRELE ȘI INDICATØRII
CARD:  XRA   A          ;TRY = 0
       STA   TRY
       SHLD  BUFR       :SALVARE ADRESĂ BUFFER IEȘIRE
CARD1: LXI   H,TCARD    ;INIȚIALIZARE ADRESĂ BUFFER
                        ;INTRARE
       MVI   E, 80      ;INIȚIALIZARE NUMĂR ØCTEȚI
       IN    IØST       ;ADU ØCTET STARE I/E
       ANI   CRRDY      ;REȚINE BITUL CRRDY
       JZ    CARD1      ;BUCLEAZĂ PÎNĂ CÎND PERIFERIC GATA
       ØUT   CRXFD      ;TRIMITE IMPULSUL EXTERNAL FEED
       IN    IØST       ;ADU ØCTET STARE I/E
       ANI   CRBSY      ;REȚINE BITUL CRBUSY
       JZ    CARD1      ;BUCLEAZĂ PÎNĂ CÎND PERIFERIC
                        :ØCUPAT
CARD2: IN    IØST       ;ADU ØCTET STARE I/E
       MØV   C, A       ;SALVEAZĂ-L ÎN REGISTRUL C
       ANI   DTRDY      ;REȚINE BITUL DATRDY
       JNZ   CARD3      ;CÎND „1" ADU DATELE
       MØV   A, C       ;CÎND „0" REFĂ ØCTETUL DE STARE
       ANI   CRRDY      ;TESTEAZĂ CRRDY
       JNZ   CARD2      ;BUCLEAZĂ DACĂ READY = „1"
CARD6: LDA   TRY        ;TRATAREA ERØRII, TEST TRY
       ØRA   A          ;PØZIȚIØNEAZĂ INDICATØRII
       JNZ   CARD1      ;CÎND TRY = 1 REIA RUTINA CARD
       INR   A          ;TRY = TRY + 1
       STA   TRY
CARD4: MVI   E, 80      ;LISTEAZĂ ULTIMA CARTELĂ
       LHLD  BUFR
CARD5: MØV   C, M       ;ADU CØLØANA CURENTĂ
       CALL  CØ         ;TIPĂREȘTE LA CØNSØLĂ
       DCR   E          ;TEST DACĂ SFÎRȘITUL CARTELEI
       JZ    CARD9      ;E GATA, BATE CR, LF LA CØNSØLĂ
       INX   H          ;NU E GATA, TIPĂREȘTE MAI DEPARTE
       JMP   CARD5
```

```
CARD9:  CALL   CRØUT   ;BATE CR, LF LA CØNSØLĂ
        JMP    CARD1   ;REIA RUTINA CARD
CARD3:  IN     CRDAT   ;PRELUAREA DATELØR ØCTETUL 1
        MØV    M,A     ;MEMØRARE ÎN BUFFER INTRARE
        INX    H       ;AVANS ADRESĂ
        IN     CRDAT   ;PRELUAREA DATELØR ØCTETUL 2
        MØV    M,A     ;MEMØRARE ÎN BUFFER INTRARE
        INX    H       ;AVANS ADRESĂ
        DCR    E       ;DECREMENTARE CØNTØR CØLØANE
        JNZ    CARD2   ;CARTELA NU E GATA, REIA
CARD7:  IN     IØST    ;CARTELA CØMPLET CITITĂ
        MØV    C,A     ;SALVEAZĂ ØCTET STARE I/E
        ANI    CRRDY   ;REȚINE CRRDY
        JZ     CARD6   ;ERØARE SINCRØNIZARE, STACKER
        MØV    A, C    ;REFĂ ØCTET STARE I/E
        ANI    CRBSY   ;REȚINE CRBUSY
        JNZ    CARD7   ;BUCLEAZĂ CÎND PERIFERIC ØCUPAT
        IN     IØST    ;PERIFERIC LIBER, TESTEAZĂ READY
        ANI    CRRDY   ;REȚINE CRRDY
        JZ     CARD6   ;ERØARE SINCRØNIZARE, STACKER
        LXI    H,TCARD;CØNVERSIE HØLLERITH-ASCII
        MVI    E,80
CARD8:  MØV    B,M     ;ADU ØCTET 1 ÎN B
        INX    H
        MØV    C, M    ;ADU ØCTET 2 ÎN C
        INX    H
        SHLD   TCRDS   ;SALVEAZĂ ADRESA BUFFER INTRARE
        PUSH   D       ;SALVEAZĂ (D, E)
        CALL   HASC    ;CHEAMĂ RUTINA DE CØNVERSIE
        PØP    D       ;REFĂ (D, E)
        LHLD   BUFR    ;ADRESA DIN BUFFER IEȘIRE
        MØV    M, A    ;MEMØREAZĂ ØCTET ÎN BUFFER
        INX    H       ;AVANS ADRESĂ BUFFER IEȘIRE
        SHLD   BUFR    ;SALVARE ADRESĂ BUFFER IEȘIRE
        DCR    E       ;DECREMENTARE CØNTØR ØCTEȚI
        RZ             ;CØNVERSIE GATA, REVENIRE
        LHLD   TCRDS   ;REPETĂ ADRESA BUFFER INTRARE
        JMP    CARD8   ;REIA CØNVERSIA
IØST    EQU    0FDH    ;ADRESA ØCTET STARE I/E
CRDAT   EQU    0EFH    ;ADRESA ØCTET DATE CITITØR CARTELE
CRXFD   EQU    0EFH    ;ADRESA CØMANDĂ IMPULS EXTERNAL
                       ;FEED
CRRDY   EQU    10H     ;MASCA PENTRU BITUL 4 = CR READY
DTRDY   EQU    20H     ;MASCA PENTRU BITUL 5 = DATA
                       ;STRØBE
CRBSY   EQU    40H     ;MASCA PENTRU BITUL 6 = CR BUSY
CØ      EQU    6BFH    ;ADRESA RUTINEI CØ (CØNSØLE
                       ;ØUTPUT)
```

```
CRØUT   EQU   6C1H      ;ADRESA RUTINEI CRØUT (CR, LF,
                        ;OUTPUT)
PTMEM   EQU   $         ;SALVAREA ADRESEI CURENTE
                        ;ASAMBLØR
MEMRY   EQU   0BE00H    ;ZØNA DE LUCRU PENTRU PERIFERICE
        ØRG   MEMRY
TRY:    DS    1         ;INDICATØRUL DE ÎNCERCĂRI DE
                        ;CITIRE
BUFR:   DS    2         ;ADRESA CURENTĂ ÎN BUFFER IEȘIRE
TCRDS:  DS    2         ;ADRESA CURENTĂ ÎN BUFFER
                        ;INTRARE
TCARD:  DS    160       ;BUFFER INTRARE — 160 ØCTEȚI
        ØRG   PTMEM     ;REFACERE ADRESA CURENTĂ
                        ;ASAMBLØR
```

Observăm că în cursul executării rutinei CARD sînt chemate alte două rutine: CØ (Console Output) care tipărește la consolă caracterul al cărui cod ASCII i se transmite prin intermediul registrului C și HASC ce convertește codul Hollerith găsit în dublul registru (B, C) în codul ASCII corespondent care e transmis la ieșirea din rutina HASC prin intermediul acumulatorului.

Rutina de conversie folosește un tabel de coduri THASC organizat conform modelului din figura 6.15*a*. Acest tabel are 59 de intrări, corespunzînd celor 59 de caractere recunoscute de rutină, iar fiecare intrare este constituită din doi octeți, reprezentînd codul Hollerith al unui caracter (fig. 6.15 *b*). Rutina HASC execută o căutare secvențială în tabel, codul rezultant (ASCII) fiind reprezentat de numărul de ordine al intrării în tabel pentru care s-a găsit coincidență, exprimat în binar, număr la care se adaugă constanta 20_{16}. În cazul cînd nu se găsește coincidență pentru nici o intrare, codul de ieșire al rutinei va fi 20H corespunzînd caracterului „blanc"; cu alte cuvinte caracterele necunoscute sînt înlocuite automat cu „blanc".

În continuare dăm secvența de program ce implementează această rutină și tabelul THASC:

```
RUTINA DE CØNVERSIE HØLLERITH — ASCII
;       INTRĂRI:    CODUL HOLLERITH AL CARACTERULUI
;                   CITIT ÎN (B, C)
;                   B = ØCTET 1 (CØL 1—8)
;                   C = ØCTET 2 (CØL 9, 12, 11, 0)
;       IEȘIRI:     CØDUL ASCII ÎN ACUMULATØR
;       DISTRUGE:   TØATE REGISTRELE ȘI
;                   INDICATØRII
HASC:  LXI  H,THASC;INIȚIALIZEAZĂ ADRESA ÎN TAB. CØN-
                   ;VERSIE
       MVI  E, 20H ;ÎN E CØDUL ASCII AL CARACTERULUI
                   ;BLANC
       MVI  D,59   ;ÎN D NUMĂRĂTØR DE INTRĂRI ÎN THASC
```

```
HASC1:  MØV   A, B      ;ADU ØCTETUL 1
        CMP   M         ;CØMPARĂ CU ØCT 1 DIN THASC
        INX   H         ;AVANS ADRESĂ PENTRU ØCTET 2
        JNZ   HASC2     ;SALT PE ≠
        MØV   A, C      ;ADU ØCTETUL 2
        CMP   M         ;CØMPARĂ CU ØCTET 2 DIN THASC
        JZ    HASC3     ;INTRARE ÎN THASC GĂSITĂ
HASC2:  INX   H         ;AVANS ADRESĂ PT. INTRAREA
                        ;URMĂTØARE
        INR   E         ;INCREMENTEAZĂ CØD ASCII
        DCR   D         ;DECREMENTEAZĂ NUMĂRĂTØR
                        ;INTRĂRI
        JNZ   HASC1     ;REIA PENTRU INTRAREA URMĂTØARE
        MVI   A,20H     ;CARACTER NECUNØSCUT, CØDUL
                        ;BLANC
        RET             ;REVENIRE ÎN PRØG. PRINCIPAL
HASC3:  MØV   A, E      ;CØDUL ASCII ÎN ACUMULATØR
        RET             ;REVENIRE ÎN PRØG. PRINCIPAL

; TABELUL DE CØNVERSIE HØLLERITH — ASCII
THASC:  DB 00H, 00H     ;HØL = BLANC, CAR = BLANC,
                        ;ASC = 20H
        DB 0C0H, 04H    ;HØL = 12—8—7, CAR= !, ASC = 21H
        DB 0C0H, 00H    ;HØL = 8—7, CAR=", ASC = 22H
        DB 84H, 00H     ;HØL = 8—3, CAR = #, ASC = 23H
        DB 84H, 02H     ;HØL = 11—8—3, CAR = $, ASC = 24H
        DB 88H, 01H     ;HØL = 0—8—4, CAR = %, ASC = 25H
        .
        .
        .
        DB 00H, 81H     ;HØL = 0—9, CAR=Z, ASC = 5AH
```

6.5. INTERFAȚA PENTRU IMPRIMANTA PARALELĂ

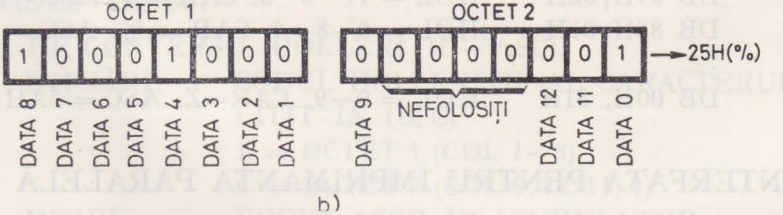
Aceasta este o interfață de ieșire paralelă pe 8 biți din cadrul SD-8080 care, împreună cu rutina de I/E specializată PRINT, permite conectarea la SD-8080 a unei imprimante paralele de 400 linii/ min. de tip RCD 508 M.

Interfața folosește următoarele adrese de I/E (vezi și tabelele 4.3 și 4.4):

1. Adresa de intrare FD_{16}; în octetul recepționat de pe această adresă, care are rolul unui octet de stare a I/E, bitul 7 = PRRDY (PRINTER READY) este atribuit stării imprimantei. Dacă acesta este „1" imprimanta este gata de lucru, în sensul că este alimentată, are hîrtie, bandă tușată, circuitele de comandă ale ciocănelelor sînt în stare de funcționare ș.a.m.d.

Caracter	Cod Hollerith	Cod ASCII	Caracter	Cod Hollerith	Cod ASCII
SP (b)	blanc	20	>	0-8-6	3E
!	12-8-7	21	?	0-8-7	3F
"	8-7	22	@	8-4	40
#	8-3	23	A	12-1	41
$	11-8-3	24	B	12-2	42
%	0-8-4	25	C	12-3	43
&	12	26	D	12-4	44
'	8-5	27	E	12-5	45
(	12-8-5	28	F	12-6	46
)	11-8-5	29	G	12-7	47
*	11-8-4	2A	H	12-8	48
+	12-8-6	2B	I	12-9	49
,	0-8-3	2C	J	11-1	4A
-	11	2D	K	11-2	4B
.	12-8-3	2E	L	11-3	4C
/	0-1	2F	M	11-4	4D
0	0	30	N	11-5	4E
1	1	31	O	11-6	4F
2	2	32	P	11-7	50
3	3	33	Q	11-8	51
4	4	34	R	11-9	52
5	5	35	S	0-2	53
6	6	36	T	0-3	54
7	7	37	U	0-4	55
8	8	38	V	0-5	56
9	9	39	W	0-6	57
:	8-2	3A	X	0-7	58
;	11-8-6	3B	Y	0-8	59
<	12-8-4	3C	Z	0-9	5A
=	8-6	3D			

a)



b)

Fig. 6.15. Tabelul conversiei din cod Hollerith în cod ASCII (a). Exemplu de reprezentare în memorie a unei intrări din tabelul de conversie THASC (b)

2. Adresa de ieșire FF_{16}; în octetul trimis pe această adresă biții 0—5 formează codul caracterului ce este memorat în bufferul intern al imprimantei în vederea tipăririi lui ulterioare. Imprimanta folosește codul EBCDIC, recunoscînd 64 de caractere codificate pe 6 biți. Bitul 6 al octetului de ieșire este fără semnificație, iar bitul 7 este folosit pentru a comanda tipărirea liniei ale cărei caractere au fost recepționate și memorate de imprimantă în ciclul de transfer de date anterior comenzii de tipărire.

Semnalele de interfață ale imprimantei, precum și semnificațiile lor sînt descrise mai jos:

1. DATA 0-DATA 5 sînt 6 semnale de date furnizate de către sistemul de dezvoltare, formînd codul EBCDIC al caracterului care este transmis imprimantei în vederea tipăririi.

2. $\overline{\text{DATA STROBE}}$ (activ jos), constituie semnalul de strob al datelor cu care sistemul de dezvoltare validează liniile de date către imprimantă.

3. CONTROL BIT este un semnal furnizat de sistemul de dezvoltare după aceleași specificații de timp ca și cei 6 biți de date. Cînd acest bit devine „1" codul de pe liniile de date este neglijat, CONTROL BIT fiind interpretat ca o comandă de tipărire a caracterelor ce au fost memorate în bufferul imprimantei în intervalul de timp scurs de la primirea anterioară a lui CONTROL BIT. O linie e formată din maximum 132 caractere tipăribile. Primul caracter al liniei nu este tipărit, iar codul reprezentat de acest caracter S este tratat de imprimantă ca un cod de salt.

Structura transferului de date cu imprimanta a fost prezentată în figura 6.16*a*. Vom nota că pentru a tipări o linie formată din N caractere, $0 \leqslant N \leqslant 132$, au loc $N + 2$ transferuri de date, cele două cicluri suplimentare fiind adăugate automat de rutina de comandă a acestui dispozitiv PRINT. Aceste cicluri sînt inițiate de PRINT la primirea caracterului CR = CARRIAGE RETURN, în primul trimițîndu-se CONTROL BIT pentru tipărirea celor N caractere ale liniei care au fost anterior transferate, iar în al doilea se transmite caracterul de salt al liniei următoare, S = 1.

Semnificația asociată codurilor de salt este dată în figura 6.16*b*. În cazul nostru se folosește un singur cod de salt, S = 1, corespunzînd saltului la linia următoare înaintea tipăririi.

4. $\overline{\text{CHARACTER REQUEST}}$ (activ jos) este semnalul prin activarea căruia imprimanta cere sistemului de dezvoltare un caracter.

Semnalul $\overline{\text{CHARACTER REQUEST}}$ este exploatat și ca semnal de stare a imprimantei, fiindcă atunci cînd este „0" el are și semnificația că perifericul întrunește toate condițiile de lucru necesare.

Transmiterea pe cablu a semnalelor de interfață cu imprimanta se face cu ajutorul circuitelor specializate de emisie/recepție pe cablu DM 8830/DM 8820.

Relațiile de timp ce guvernează interfața sînt prezentate în figura 6.17. După cum se vede, inițiativa aparține imprimantei care activează semnalul $\overline{\text{CHARACTER REQUEST}}$ atunci cînd aceasta este gata să primească un nou caracter. Ca răspuns, sistemul de dezvoltare poziționează liniile de date, după care activează semnalul de validare a datelor, $\overline{\text{DATA STROBE}}$. La primirea acestuia din urmă imprimanta efectuează memorarea codului caracterului după care semnalează terminarea activității sale prin punerea la „1" a lui $\overline{\text{CHARACTER REQUEST}}$. În acest moment sistemul de dezvoltare va putea executa dezactivarea liniilor de date și a validării lor, $\overline{\text{DATA STROBE}}$.

a)

Fig. 6.16. Structura transferului de date cu imprimanta (a). Semnificația codurilor de salt recunoscute de imprimantă (b)

b)

BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	SEMNIFICAȚIE
0	X	X	X	0	0	Supraimprimare
0	X	X	X	0	1	Salt 1 linie
0	X	X	X	1	0	Salt 2 linii
0	X	X	X	1	1	Salt 3 linii
1	X	0	0	0	0	Salt comandat de canalul 1 al benzii pilot
1	X	0	0	0	1	Salt comandat de canalul 2 al benzii pilot
1	X	0	0	0	1	Salt comandat de canalul 3 al benzii pilot
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	X	1	0	1	1	Salt comandat de canalul 12 al benzii pilot

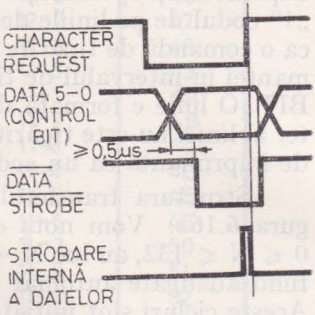


Fig. 6.17. Cronograma semnalelor de interfață ale imprimantei

Cele trei restricții de timp care guvernează dialogul între imprimantă și sistemul de dezvoltare sînt:

— $\overline{\text{CHARACTER REQUEST}}$ și datele trebuie să fie stabile cu cel puțin 0,5 μs înainte de generarea lui $\overline{\text{DATA STROBE}}$;

— $\overline{\text{DATA STROBE}}$ nu va fi dezactivat decît după trecerea în „1" a lui $\overline{\text{CHARACTER REQUEST}}$;

— liniile de date vor fi dezactivate în același timp sau mai tîrziu decît semnalul de validare $\overline{\text{DATA STROBE}}$.

În figura 6.18*a* este prezentată schema interfeței pentru imprimanta paralelă al cărei element cheie este, după cum se vede, un circuit 8212 conectat în mod de IEȘIRE (MD = „1"), ce memorează octetul primit pe magistrala de date a sistemului în timpul instrucțiunii ØUT 0FFH, moment în care circuitul 8212 este selectat (fig. 6.18*b*). Ieșirile lui 8212, $DATA_0$-$DATA_5$ comandă direct circuitele 8830 de emisie pe cablul ce leagă imprimanta de sistemul de dezvoltare. Bitul 7 al magistralei de date corespunde liniei CONTROL BIT cu ajutorul căreia se comandă tipărirea rîndului format din caracterele primite anterior acestei comenzi. Așa cum rezultă și din figura 6.18*b* în timpul selectării lui 8212 linia $\overline{\text{GOPRINT}}$ conectată la ieșirea sa $\overline{\text{INT}}$ devine „0" pentru 500 ns, revenirea ei la „1" producînd poziționarea bistabilului de tip 7474 DATA STROBE care este atacat de $\overline{\text{GOPRINT}}$ pe intrarea de

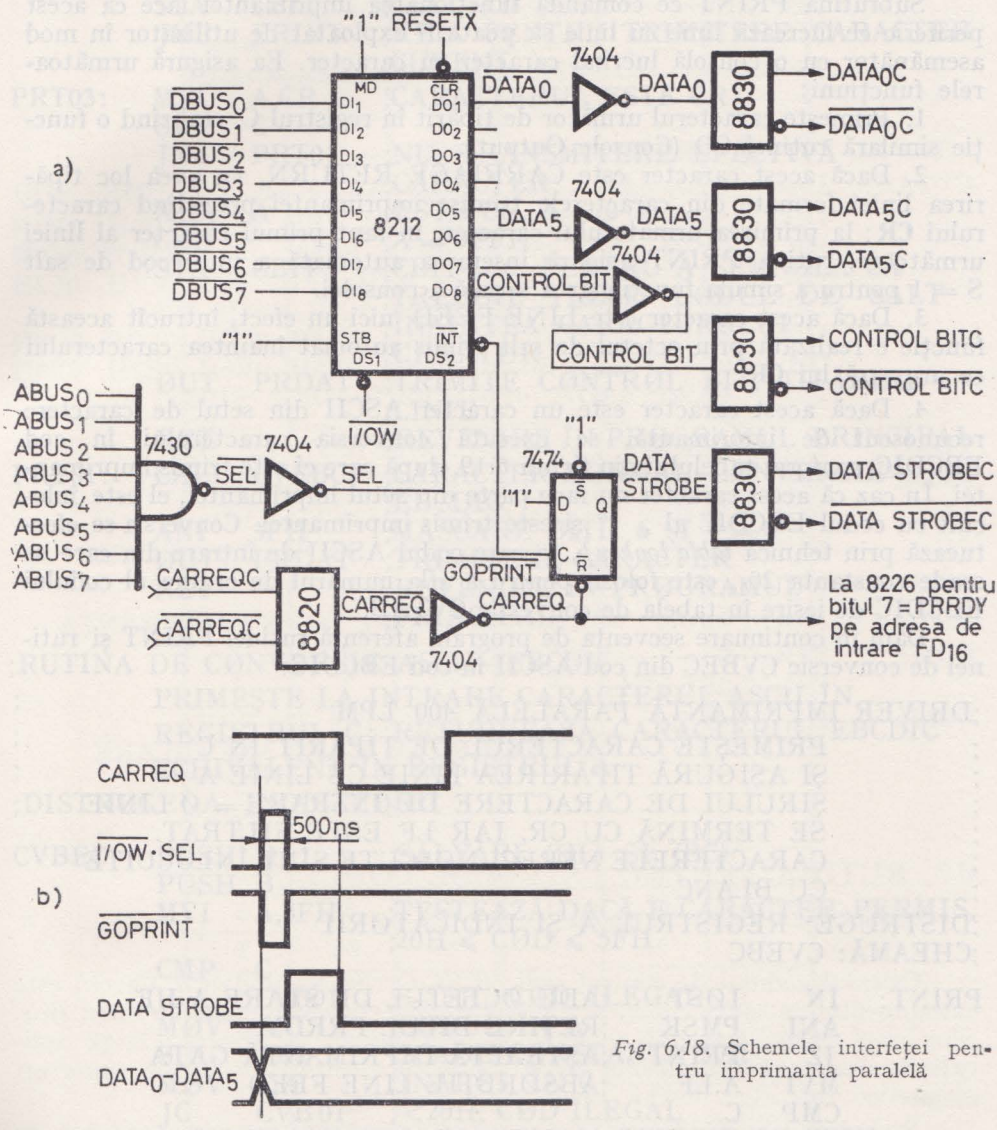


Fig. 6.18. Schemele interfeței pentru imprimanta paralelă

ceas activă pe frontul pozitiv. Se asigură în acest fel intervalul minim de stabilire a liniilor de date înaintea apariției strobului prezentat în figura 6.17. Căderea lui DATA STROBE are loc în urma căderii lui CARREQ, semnificînd faptul că datele au fost preluate de imprimantă. De asemenea, așa cum am mai amintit, CARREQ este folosit și ca semnal de stare a imprimantei, fiind transmis pe bitul 7 al octetului de stare a I/E cu ajutorul unui circuit 8226 similar cu cel din figura 6.3.

Subrutina PRINT ce comandă funcționarea imprimantei face ca acest periferic ce lucrează linie cu linie să poate fi exploatat de utilizator în mod asemănător cu o consolă lucrînd caracter cu caracter. Ea asigură următoarele funcțiuni:

1. Primește caracterul următor de tipărit în registrul C, realizînd o funcție similară rutinei CO (Console Output).

2. Dacă acest caracter este CARRIAGE RETURN, va avea loc tipărirea liniei formate din caracterele trimise imprimantei precedînd caracterului **CR**; la primirea următorului caracter, în fapt primul caracter al liniei următoare, rutina PRINT asigură inserarea automată a unui cod de salt S = 1 pentru a simula funcționarea serială a consolei.

3. Dacă acest caracter este LINE FEED, nici un efect, întrucît această funcție e realizată prin octetul de salt trimis automat înaintea caracterului ce urmează lui **CR**.

4. Dacă acest caracter este un caracter ASCII din setul de caractere recunoscut de imprimantă, se execută conversia caracterului în cod EBCDIC conform tabelului din figura 6.19, după care el este trimis imprimantei. În caz că acest caracter nu face parte din setul imprimantei, el este înlocuit cu codul EBCDIC al „ " și este trimis imprimantei. Conversia se efectuează prin tehnica *table look-up*, în care codul ASCII de intrare din care se scade constanta 20_{16} este folosit pentru a afla numărul de ordine al codului EBCDIC de ieșire în tabela de conversie CVTAB.

Dăm în continuare secvența de program aferentă rutinei PRINT și rutinei de conversie CVBEC din cod ASCII în cod EBCDIC:

```
;DRIVER IMPRIMANTĂ PARALELĂ 400 LPM
;          PRIMEȘTE CARACTERUL DE TIPĂRIT ÎN C
;          ȘI ASIGURĂ TIPĂRIREA LINIE CU LINIE A
;          ȘIRULUI DE CARACTERE DE INTRARE = Ø LINIE
;          SE TERMINĂ CU CR, IAR LF ESTE FILTRAT.
;          CARACTERELE NERECUNØSCUTE SÎNT ÎNLØCUITE
;          CU BLANC
;DISTRUGE: REGISTRUL A ȘI INDICATØRII
;CHEAMĂ: CVEBC

PRINT:  IN      IØST    ;ADU ØCTETUL DE STARE A I/E
        ANI     PMSK    ;REȚINE BITUL PRRDY
        JZ      PRINT   ;AȘTEAPTĂ IMPRIMANTĂ GATA
        MVI     A,LF    ;ABSØRBȚIE LINE FEED
        CMP     C
        RZ              ;LINE FEED, NICI UN EFECT
        LDA     CBIT    ;VEZI DACĂ ANTERIØR CR TESTÎND
                        ;CBIT
        ØRA     A
        JZ      PRT03   ;CBIT = 0, NU A FØST CR
        MVI     A,SKIP  ;CBIT ≠ 0, DECI TRIMITE CØD SALT
        ØUT     PRDAT
        XRA     A       ;A = 0
```

```
        STA   CBIT     ;CBIT = 0
        JMP   PRINT    ;REIA PENTRU TRIMITERE CARACTER
                       ;CURENT
PRT03:  MVI   A,CR     ;CARACTERUL ESTE CR?
        CMP   C
        JNZ   PRT01    ;NU, TRANSMITERE EFECTIVĂ
                       ;CARACTER
        XRA   A        ;ESTE CR, A = 0
        CMA            ;A = 0FFH
        STA   CBIT     ;CBIT ≠ 0, PENTRU A SEMNALA CĂ
                       ;TREBUIE TRIMIS CØDUL DE SALT
                       ;ÎNAINTEA CARACTERULUI
                       ;URMĂTØR
        ØUT   PRDAT    ;TRIMITE CØNTRØL BIT, TIPĂRIRE
                       ;LINIE
        RET            ;REVENIRE ÎN PRØGRAMUL PRINCIPAL
PRT01:  CALL  CVEBC    ;CARACTER NØRMAL, CØNVERSIE
                       ;EBCDIC
        ANI   3FH      ;MASCARE BIȚII 6 ȘI 7 = 0
        ØUT   PRDAT    ;TRIMITERE CARACTER
        RET            ;REVENIRE ÎN PRØGRAMUL
                       ;PRINCIPAL

;RUTINA DE CØNVERSIE ASCII—EBCDIC
;       PRIMEȘTE LA INTRARE CARACTERUL ASCII ÎN
;       REGISTRUL C; RETURNEAZĂ CARACTERUL EBCDIC
;       ECHIVALENT ÎN REGISTRUL A
;DISTRUGE: A, INDICATØRII

CVBEC:  PUSH  H        ;SALVĂRI (HL) ȘI (BC)
        PUSH  B
        MVI   A,5FH    ;TESTEAZĂ DACĂ E CARACTER PERMIS,
                       ;20H ⩽ CØD ⩽ 5FH
        CMP   C
        JC    CVB01    ;>5FH, CØD ILEGAL
        MØV   A,C      ;CØDUL ÎN A
        SUI   20H      ;SCADE CØNSTANTA 20H
        MØV   C,A      ;ÎNAPØI, ÎN C
        JC    CVB01    ;<20H, CØD ILEGAL
        LXI   H,CVTAB  ;CAZ NØRMAL, CØNVERSIE PRIN
                       ;TABEL
        MVI   B,0      ;B = 0
        DAD   B        ;ADRESA ÎN CVTAB
        MØV   A,M      ;EXTRAGE CØDUL
        JMP   CVB01+2  ;TERMINARE CØNVERSIE
CVB01:  MVI   A,40H    ;CAR. NECUNØSCUT, ÎNLØCUIT CU
                       ;BLANC
```

```
        PØP   B          ;REFACERE (HL) ŞI (BC)
        PØP   H
        RET              ;REVENIRE ÎN PRØGRAMUL
                         ;PRINCIPAL

;TABEL DE CØNVERSIE ASCII-EBCDIC
CVTAB:  DB    40H, 5AH, 7FH, 7BH, 5BH, 6CH, 50H, 7DH, 4DH
        DB    5DH, 5CH, 4EH, 6BH, 60H, 4BH, 61H, 0F0H, 0F1H
        DB    0F2H, 0F3H, 0F4H, 0F5H, 0F6H, 0F7H, 0F8H, 0F9H
        DB    7AH, 5EH, 4CH, 7EH, 6EH, 6FH, 7CH, 0C1H, 0C2H
        DB    0C3H, 0C4H, 0C5H, 0C6H, 0C7H, 0C8H, 0C9H
        DB    0D1H, 0D2H, 0D3H, 0D4H, 0D5H, 0D6H, 0D7H, 0D8H
        DB    0D9H, 0E2H, 0E3H, 0E4H, 0E5H, 0E6H, 0E7H, 0E8H
        DB    0E9H, 0ADH, 0E0H, 0BDH, 5FH, 6DH

IØST    EQU   0FDH       ;ADRESA ØCTET STARE I/E
PMSK    EQU   80H        ;MASCA PT. BITUL 7 = PRRDY
LF      EQU   0AH        ;CARACTERUL LINE FEED
CR      EQU   0DH        ;CARACTERUL CARRIAGE RETURN
SKIP    EQU   1          ;CØDUL DE SALT
PRDAT   EQU   0FFH       ;ADRESA IEŞIRE DATE IMPRIMANTA
CRNT    EQU   $          ;ADRESA CURENTĂ
        ØRG   0BEA5H
CBIT:   DB    0FFH       ;REZERVARE CBIT ÎN RAM
        ØRG   CRNT       ;REFACERE ADRESĂ ASAMBLARE
```

6.6. RUTINA PENTRU CALCULUL CODULUI DE CONTROL CICLIC

Calculul codului de control ciclic (CRC-Cyclic Redundancy Check) este actualmente cea mai răspîndită metodă de detectare a erorilor în cazul şirurilor lungi de biţi transmişi în serie. Această metodă îşi găseşte o largă aplicaţie în domeniul transmisiilor de date şi în cel al înregistrărilor pe suporturi magnetice: disc, disc flexibil, casetă magnetică etc.

Probabilitatea de detectare a erorilor depinde de lungimea codului de control folosit şi de lungimea mesajului. Se folosesc în mod curent următoarele polinoame generatoare:

$$P(x) = x^{16} + x^{15} + x^2 + 1 \quad \text{(CRC 16 Forward)}$$
$$P(x) = x^{16} + x^{11} + x + 1 \quad \text{(CRC 16 Reverse)}$$
$$P(x) = x^{16} + x^{12} + x^5 + 1 \quad \text{(CCITT Forward)}$$
$$P(x) = x^{16} + x^{11} + x^4 + 1 \quad \text{(CCITT Reverse)}$$

Caracter	Cod ASCII	Cod EBCDIC	Caracter	Cod ASCII	Cod EBCDIC
b	20	40	@	40	7C
!	21	5A	A	41	C1
"	22	7F	B	42	C2
#	23	7B	C	43	C3
$	24	5B	D	44	C4
%	25	6C	E	45	C5
&	26	50	F	46	C6
'	27	7D	G	47	C7
(	28	4D	H	48	C8
)	29	5D	I	49	C9
*	2A	5C	J	4A	D1
–	2B	4E	K	4B	D2
,	2C	6B	L	4C	D3
-	2D	60	M	4D	D4
.	2E	4B	N	4E	D5
/	2F	61	O	4F	D6
0	30	F0	P	50	D7
1	31	F1	Q	51	D8
2	32	F2	R	52	D9
3	33	F3	S	53	E2
4	34	F4	T	54	E3
5	35	F5	U	55	E4
6	36	F6	V	56	E5
7	37	F7	W	57	E6
8	38	F8	X	58	E7
9	39	F9	Y	59	E8
:	3A	7A	Z	5A	E9
;	3B	5E	[	5B	AD
<	3C	4C	\	5C	E0
=	3D	7E	]	5D	BD
>	3E	6E	∧	5E	5F
?	3F	6F	—	5F	6D

Fig. 6.19. Tabela conversiei din cod ASCII în cod EBCDIC

Considerînd mesajul de transmis ca o înșiruire de n biți:

$$a_{n-1}a_{n-2} \dots a_1a_0,$$

el se va transcrie sub forma unui polinom de grad $n - 1$:

$$M(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0.$$

Considerăm că mesajul este completat în partea biților săi cei mai puțin semnificativi cu 16 zerouri, număr corespunzînd gradului polinomului generator și numărului de coeficienți (biți) ai cîtului împărțirii cu $P(x)$. Obținem astfel mesajul:

$$a_{n-1}a_{n-2} \dots a_1a_0 \underbrace{00 \dots 0}_{16 \text{ ori}}$$

căruia îi corespunde polinomul $x^{16} \cdot M(x)$. Se face acum împărțirea $x^{16} \cdot M(x) : P(x)$, obținîndu-se cîtul $Q(x)$, care nu ne interesează și restul $R(x)$ de grad 15. Avem deci:

$$x^{16} \cdot M(x) \equiv Q(x) \cdot P(x) + R(x), \text{ unde}$$

$$R(x) = r_{15}x^{15} + r_{14}x^{14} + \dots + r_0$$

Adunăm restul la polinomul deplasat $x^{16} \cdot M(x)$ și obținem polinomul

$$T(x) = x^{16} \cdot M(x) + R(x),$$

căruia îi corespunde următorul mesaj de $n + 16$ biți:

$$\underbrace{a_{n-1}a_{n-2} \dots a_1a_0}_{\text{mesajul}} \quad \underbrace{r_{15}r_{14} \dots r_0}_{\text{codul de control}}$$

Acesta este mesajul trimis pe linia de transmisie de date ce sau înregistrat pe suportul magnetic; el este format după cum se vede din mesajul propriu-zis, căruia i se adaugă codul de control CRC format din cei 16 biți ai restului.

La recepție mesajul $T(x)$ e divizat cu $P(x)$; în cazul recepției corecte restul obținut e nul, în caz contrar mesajul recepționat $T(x)$ a fost alterat de un semnal de eroare care se manifestă sub forma unui polinom $E(x)$, unde $T(x) = M(x) + E(x)$. Există o anumită probabilitate ca $T(x)$ să fie la rîndul lui divizibil cu P(x), care este de fapt probabilitatea de nedetectare a unui mesaj eronat. Aceasta are loc pentru cazul cînd $E(x)$ este și el divizibil cu $P(x)$; în literatură se indică faptul că probabilitatea de apariție a unei erori nedetectabile folosind un cod ciclic bazat pe un polinom generator de gradul 16 este:

— $P = 0$ dacă $n \leqslant 16$, cu alte cuvinte nu pot exista erori nedetectabile dacă șirul de transmis e mai scurt decît codul de control deoarece, evident, în acest caz $E(x)$ nu poate fi divizibil cu $P(x)$

— $P = \dfrac{2^{n-16} - 1}{2^n - 1}$ dacă $n > 16$, valoare care se apropie de 2^{-16} pentru n foarte mare. Altfel spus, probabilitatea globală de detectare a unei erori prin această metodă este 0,99998.

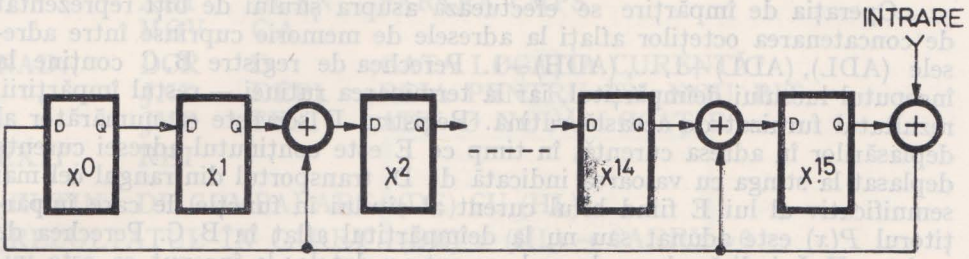


Fig. 6.20. Circuitul de calcul al CRC pentru polinomul $P(x) = x^{16} + x^{15} + x^2 + 1$

S-a demonstrat, de asemenea, că metoda detectează cu certitudine un singur bit eronat, iar unica secvență nedetectabilă ce conține 2 biți eronați este aceea în care biții greșiți sînt separați prin $2^n - 1$ zerouri [18].

Divizibilitatea lui $T(x)$ cu $P(x)$ în cazul recepției corecte e evidentă. Într-adevăr se obține:

$$T(x) \equiv x^{16} \cdot M(x) + R(x) \equiv Q(x) \cdot P(x) + R(x) + R(x) \equiv Q(x) \cdot P(x),$$

deoarece sumele sînt considerate modulo 2.

Implementarea hardware a calculului codului de control se face cu ajutorul unui registru de deplasare cu reacție de tipul celui prezentat în figura 6.20.

În cele ce urmează vom prezenta funcționarea unui program de calcul al CRC care poate suplini funcția circuitului de deplasare cu reacție, în cazul sistemelor bazate pe microprocesor.

Rutina efectuează calculul CRC considerînd ca lanț de intrare datele cuprinse între adresele de memorie ADL și ADH, care trebuie pregătite de utilizator înainte de apelul acestei rutine. Polinomul generator folosit este transmis rutinei cu ajutorul zonei de 2 octeți PGEN; în figura 6.21 a fost ilustrat modul de completare al acestei zone pentru diferite tipuri de polinoame generatoare.

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	PGEN = 8005H
	$P(x) = x^{16} + x^{15} + x^2 + 1$																(CRC 16 Forward)
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	PGEN = 4003H
	$P(x) = x^{16} + x^{14} + x + 1$																(CRC 16 Reverse)
1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	PGEN = 1021H
	$P(x) = x^{16} + x^{12} + x^5 + 1$																(CCITT Forward)
1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	PGEN = 0811H
	$P(x) = x^{16} + x^{11} + x^4 + 1$																(CCITT Reverse)

Fig. 6.21. Completarea zonei PGEN pentru diferite tipuri de polinoame generatoare

Operația de împărțire se efectuează asupra șirului de biți reprezentat de concatenarea octeților aflați la adresele de memorie cuprinse între adresele (ADL), (ADL)+1, ..., (ADH)—1. Perechea de registre B, C conține la începutul lucrului deîmpărțitul, iar la terminarea rutinei — restul împărțirii, rezultatul furnizat de această rutină. Registrul D servește ca numărător al deplasărilor în adresa curentă, în timp ce E este conținutul adresei curente deplasat la stînga cu valoarea indicată de D, transportul din rangul cel mai semnificativ al lui E fiind bitul curent al cîtului în funcție de care împărțitorul $P(x)$ este adunat sau nu la deîmpărțitul aflat în B, C. Perechea de registre H, L indică adresa de unde se extrag datele; la început ea este inițializată la valoarea (ADL). Organigrama rutinei de calcul al CRC este dată în figura 6.22. Etichetele din dreptul căsuțelor de pe organigramă corespund etichetelor din programul prezentat mai jos.

```
;RUTINA DE CALCUL AL CRC
;REZULTATUL E FURNIZAT ÎN B, C
CRC:    LHLD  ADL      ;ADRESA PRIMEI LOCAȚII
        MØV   A,M      ;ÎNCARCĂ DEÎMPĂRȚITUL ȘI ADUNĂ
        XRI   RIN SHR 8      ;RESTUL INIȚIAL
        MØV   B,A
        INX   H
        MØV   A,M
        XRI   RIN
        MØV   C,A
TEST:   XCHG                 ;SALVEAZĂ ULTIMA ADRESĂ ÎN DE
        LHLD  ADH            ;ADRESA FINALĂ
        CALL  DCMP           ;CØMPARĂ ADRESELE
        JNC   EXIT           ;TEST TERMINARE
        XCHG                 ;REFĂ ADRESA VECHE
        INX   H              ;ADU LØCAȚIA CURENTĂ
        MØV   E,M
        MVI   D,8            ;INIȚIALIZ. NUM. DEPLASĂRI
DEPL:   MØV   A,E            ;DEPLASARE STÎNGA DEÎMPĂRȚIT
        RAL                  ;ȘI LØCAȚIA CURENTĂ LEGATE
        MØV   E,A
        MØV   A,C
        RAL
        MØV   C,A
        MØV   A,B
        RAL
        MØV   B,A            ;B7→CARRY
        JNC   NAD            ;TEST BITUL CMS DEÎMPĂRȚIT
        MØV   A,B            ;ADUNARE ÎMPĂRȚITØR
        XRI   PGEN SHR 8; PARTEA CMS
        MØV   B,A
        MØV   A,C
```

```
        XRI   PGEN  ;PARTEA CMPS
        MØV   C,A
NAD:    DCR   D     ;GATA LØCAȚIA CURENTĂ?
        JNZ   DEPL  ;REIA PENTRU UN NØU BIT
        JMP   TEST  ;ADU Ø NØUĂ LØCAȚIE
EXIT:   RET
;RUTINA DE CØMPARARE (DE) CU (HL)
;REZULTATUL ÎN CARRY: (DE) ⩾ (HL) → CARRY=0
;                    (DE) < (HL) → CARRY=1
DCMP:   MØV   A,E
        SUB   L
        MØV   A,D
        SBB   H
        RET
ADL:    DS    2     ;ZØNA ADRESĂ INIȚIALĂ
ADH:    DS    2     ;ZØNA ADRESĂ FINALĂ
; DIFERITE TIPURI DE PØLINØAME GENERATØARE
PGEN:   DW    8005H ;CRC 16 FORWARD
        DW    4003H ;CRC 16 REVERSE
        DW    1021H ;CCITT FORWARD
        DW    0811H ;CCITT REVERSE
; RESTUL INIȚIAL
RIN:    DW          ;0FFFFH; PENTRU CCITT X.25
        DW    0     ;PENTRU ALTE PRØCEDURI
```

În încheierea acestui paragraf va trebui să menționăm că în standardele de transmisie de date sînt indicate și alte metode de calcul al codului de control. De exemplu, în cadrul procedurii CCITT X.25 [19] codul de control FCS (Frame Check Sequence) este complementul față de 1 al restului obținut prin divizarea mesajului la polinomul generator $P(x) = x^{16} + x^{12} + x^5 + 1$ (CCITT Forward), restul inițial fiind fixat la 1 peste tot. La recepție, restul inițial e de asemenea fixat la 1 peste tot, mesajul urmat de FCS se divide prin același polinom generator, obținîndu-se în caz de transmisie corectă restul 0001 1101 0000 11111.

Rutina prezentată poate fi cu ușurință adaptată pentru a corespunde acestor cerințe dacă restul inițial RIN e fixat la 0FFFFH, făcîndu-se sumă modulo 2 cu primele două locații ale deîmpărțitului (mesajului). De asemenea, mesajului emis i se vor adăuga 2 octeți nuli. În fine, restul obținut va trebui complementat la terminarea împărțirii, la ieșirea din rutină. Pentru recepție va trebui luat în considerare numai amendamentul care se referă la fixarea restului inițial RIN la 0FFFFH și la includerea în zona de lucru a rutinei a octeților ce cuprind FCS (ultimii doi octeți care la emisie au fost nuli).

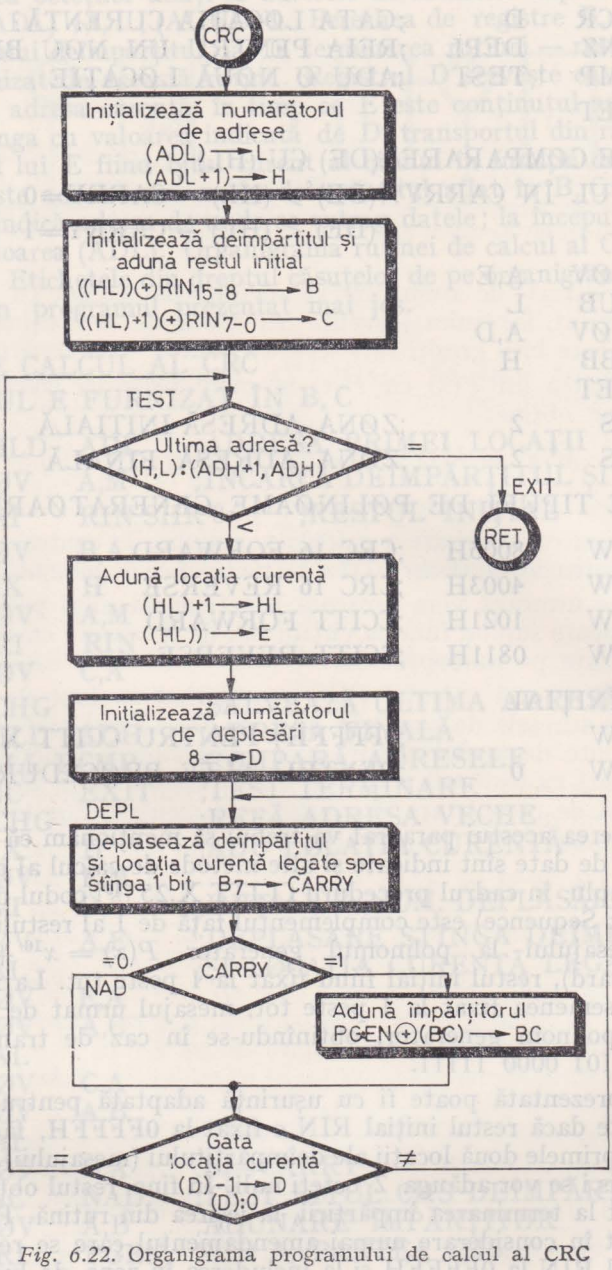


Fig. 6.22. Organigrama programului de calcul al CRC

6.7. GENERATORUL VITEZELOR DE TELETRANSMISIE

Generatorul vitezelor de teletransmisie are rolul de a furniza semnalele de ceas necesare funcționării interfețelor-serie, consolă și modem, din cadrul SD-8080.

Ceasul interfeței pentru consolă SERCLK trebuie să permită funcționarea programată a acesteia pe una dintre vitezele: 110 Bd, 200 Bd, 300 Bd sau 600 Bd. El trebuie să aibă o frecvență de 7 ori mai mare decît viteza de lucru și este derivat din ceasul stabilizat cu cuarț Ø1 al sistemului, cu frecvența de 2MHz. În figura 6.23 sînt menționate constantele de divizare și precizia obținută în acest fel; constatăm că eroarea relativă nu depășește în cel mai rău caz +0,04%.

Viteza de lucru (Bd)	Frecvența SERCLK (Hz)	Constanta de divizare	Eroarea relativă (%)
110	770	2597	+0,0155
200	1400	1428	+0,0386
300	2100	952	+0,004
600	4200	476	+0,004

Fig. 6.23. Constantele de divizare pentru realizarea semnalului de ceas al interfeței pentru consolă.

În figura 6.24 este prezentat divizorul de frecvență programabil destinat să furnizeze ceasul SERCLK. Elementul principal al schemei este numărătorul binar cu reacție de ștergere, format din trei circuite 7493, acționat la intrare cu semnalul de ceas Ø1. Reacția de ștergere este realizată cu ajutorul bistabilului XDCLR, a cărui funcție de intrare S/XDCLR este ieșirea unui multiplexor comandat de semnalele de programare TS_0 și TS_1. Numărătorul evoluează liber din starea inițială 0 pînă în starea **P**, cînd apare condiția de intrare S/XDCLR a bistabilului de ștergere a întregului numărător. Starea imediat următoare **P+1** a numărătorului este înlocuită cu starea 0, din cauza poziționării lui XDCLR care este „1" pe parcursul acestei stări. La tranziția următoare a lui Ø1, numărătorul rămîne în starea 0 pe baza condiției de forțare la „0", care e prezentă pe timpul tranziției ceasului, după care XDCLR devine „0". Concluzia este că numărătorul realizează o divizare cu P+2 dacă funcția de reacție S/XDCLR apare în starea **P**. Pentru obținerea lui SERCLK, se folosesc trei funcții de reacție diferite după cum urmează: S/XDCLR= = **2595** pentru realizarea divizării cu 2597, S/XDCLR=**1427** pentru divizarea cu 1429 și S/XDCLR=**474** pentru divizarea cu 476. Semnalul SERCLK este, la rîndul lui, ieșirea unui multiplexor, ale cărui intrări de selecție sînt semnalele de programare TS_0 și TS_1. Ceasul pentru viteza de 300 Bd este obținut prin divizarea cu 2 a ceasului pentru 600 Bd, ambelor frecvențe corespunzîndu-le aceeași constantă de divizare.

Semnalele TS_0 și TS_1 sînt ieșirile unor bistabili, care sînt poziționați cu ajutorul codului de pe biții 0 și respectiv 1 ai magistralei de date, în timpul

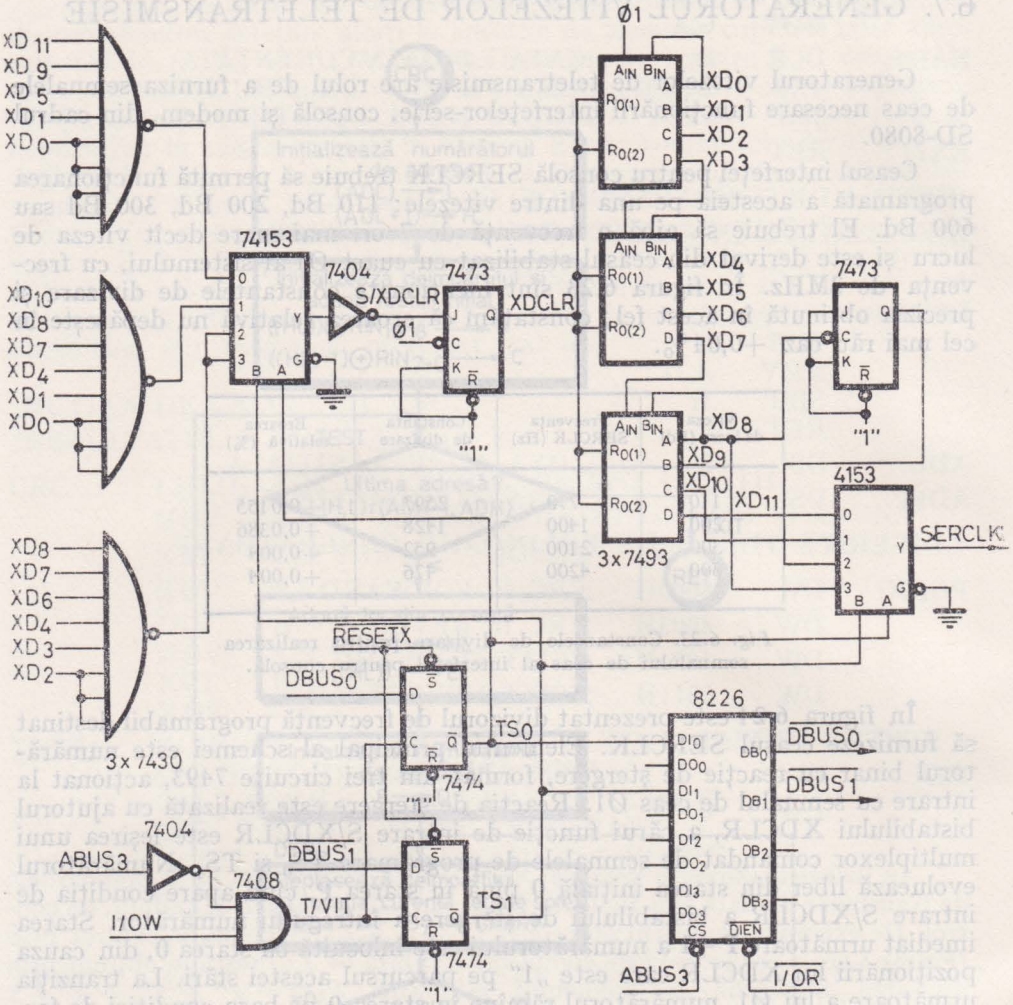


Fig. 6.24. Schemele generatorului de ceas programabil pentru consolă.

executării instrucțiunii ØUT 0F7H, pe ceasul $\overline{ABUS_3} \cdot I/OW$; aceste semnale pot fi testate de procesor pe aceiași biți ai octetului citit de la adresa de intrare $F7_{16}$, lucru pentru care se folosește un circuit 8226 selectat cu $\overline{ABUS_3} \cdot I/OR$. Cele patru combinații de cod posibile corespund celor patru viteze de funcționare ale interfeței pentru consolă, după cum rezultă din tabelele 4.3 și 4.4.

Ceasul interfeței pentru modem TRCMOD este necesar să permită funcționarea programată a interfeței pe una dintre vitezele: 110 Bd, 200 Bd, 300 Bd, 600 Bd sau 1200 Bd. El trebuie să aibă o frecvență de 16 ori mai mare ca frecvența de lucru, deoarece circuitul 8251 care implementează interfața este programat să primească la intrările TXC și RXC o frecvență

egală cu de 16 ori frecvența de lucru. Întrucît în schemă, înainte de intrările de ceas ale circuitului 8251, este montat un bistabil avînd rolul de a simetriza impulsurile S/110, S/200, S/300 S/600 și S/1200 generate de divizorul de frecvență acționat cu ceasul Ø1, rezultă că frecvențele acestora sînt un multiplu de 32 al frecvenței de lucru.

În tabelul din figura 6.25 sînt prezentate constantele de divizare și erorile relative obținute.

Viteza de lucru (Bd)	Semnalul / Frecvența (KHz)	Constanta de divizare	Eroarea relativă (%)
110	$\frac{S/110}{3,52}$	$13 \times 4 \times 11 = 572$	−0,67
200	$\frac{S/200}{6,4}$	$13 \times 8 \times 3 = 312$	+0,16
300	$\frac{S/300}{9,6}$	$13 \times 16 = 208$	+0,16
600	$\frac{S/600}{19,2}$	$13 \times 8 = 104$	+0,16
1200	$\frac{S/1200}{38,4}$	$13 \times 4 = 52$	+0,16

Fig. 6.25. Constantele de divizare pentru realizarea semnalului de ceas ale interfeței pentru modem

În figura 6.26 este prezentată schema divizorului de frecvență programabil destinat să furnizeze semnalul TRCMOD. El este constituit din mai multe numărătoare binare sincrone de tip 74161, cu reacție locală cascadată, dintr-un *latch* ce poate fi încărcat prin program pe adresa $F7_{16}$ și, în fine, dintr-un multiplexor comandat pe intrările de selecție cu ieșirile registrului programabil, a cărui ieșire după divizarea cu 2 reprezintă ceasul TRCMOD. Avantajul acestui tip de schemă, cu numărătoare cu reacție locală cascadată, în raport cu cea cu numărător cu reacție globală folosită pentru implementarea ceasului pentru consolă, este o implementare mai simplă, contrabalansată însă de o eroare relativă mai mare.

Ca exemplu de utilizare, dăm mai jos o secvență de program concepută ca o rutină, care în urma dialogului cu utilizatorul schimbă viteza de lucru a modemului în funcție de cifra tastată de acesta la consolă, după cum urmează:

— 0 pentru 110 Bd
— 1 pentru 200 Bd
— 2 pentru 300 Bd
— 3 pentru 600 Bd
— 4 pentru 1200 Bd

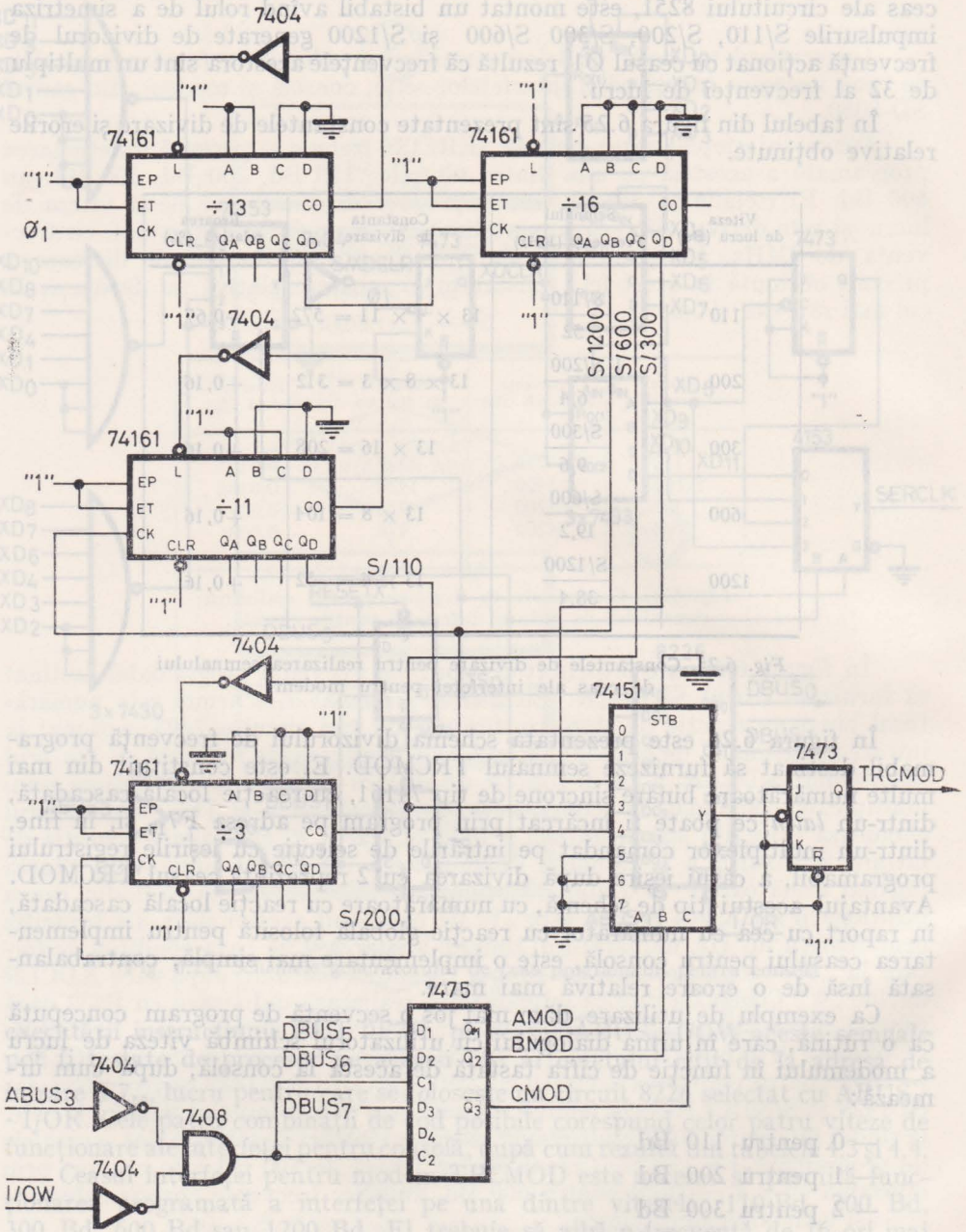


Fig. 6.26. Schemele generatorului de ceas programabil pentru modem

```
; STABILIREA VITEZEI DE LUCRU PENTRU MØDEM
VMØD:   CALL    CRØUT ;CR, LF LA CØNSØLA
        MVI     C,64H   ;TIPĂREȘTE F=
        CALL    CØ
        MVI     C,3DH
        CALL    CØ
        CALL    CI      ;ADU CIFRA DE LA CØNSØLĂ
        MØV     C,A     ;SCRI-Ø ÎN ECØU
        CALL    CØ
        CPI     30H     ;CØNTRØL VALIDITATE
        JC      ERRØR   ;CØD<30H (ZERØ)
        CPI     35H
        JNC     ERRØR   ;CØD≥35H (CINCI)
        RLC             ;DEPLASEAZĂ BIȚII CODULUI
        RLC             ;ÎN PØZIȚIILE 7, 6, 5
        RLC
        RLC
        RLC
        ØUT     MØDVT ;CØMANDĂ DIVIZØR
        RET
ERRØR:  MVI     C,20H   ;BLANC
        CALL    CØ
        MVI     C,3FH   ;„?"
        CALL    CØ
        JMP     VMØD    ;REIA ÎNTREBAREA
MØDVT   EQU     0F7H    ;ADRESA DIVIZØRULUI
CRØUT   EQU     444H    ;RUTINA CR,LF
CØ      EQU     6BFH    ;RUTINA CØNSØLE ØUTPUT
CI      EQU     6B5H    ;RUTINA CØNSØLE INPUT
```

6.8. INTERFAȚA PENTRU DOUĂ UNITĂȚI DE CASETE MAGNETICE

6.8.1. GENERALITĂȚI

În acest paragraf, ne propunem să descriem construcția și funcționarea interfeței pentru casete magnetice din cadrul SD-8080.

Interfața permite conectarea alternată la sistemul de dezvoltare a două unități de casete magnetice de tip MFE 250B. Caracteristica distinctivă a acestui tip de derulor este că el realizează viteza constantă de defilare a benzii prin fața capului de citire/scriere pe baza semnalului preînregistrat pe cea de-a doua pistă a casetei, de sincronizare, semnal cu o spațiere constantă de 1600 fci (flux changes per inch). Acest principiu a permis construirea unui derulor foarte robust din punct de vedere mecanic.

Densitatea de înregistrare maxim admisă pe casete este de 800 biți/inch, ceea ce se traduce printr-un debit de transfer de 1000 oct/s la viteza de derulare normală de 10 inch/s.

Lungimea benzii din casetă fiind de aproximativ 90 m (300 ft), aceasta corespunde unei capacități teoretice maxime de 283,5 Kocteți.

Informația este structurată în fișiere care se compun la rîndul lor din blocuri. Un fișier poate fi recunoscut cu ajutorul blocului său de identificare (*header*), format din patru caractere ASCII numerice urmate de doi octeți conținînd o informație de control de tip CRC (vezi § 6.6). Blocul de identificare este întotdeauna primul bloc al fișierului. Pentru sincronizarea citirii sale, orice bloc debutează cu un caracter special numit preambul, avînd codul hexazecimal 4E, care nu este luat în considerare la calculul CRC. După blocul de identificare urmează o porțiune de bandă polarizată uniform din punct de vedere magnetic, deci neînregistrată, numită spațiu interbloc (*gap*), care are rolul de a separa fizic blocurile între ele, permițînd unității de casetă să efectueze în acest spațiu un ciclu de frînare completă și demarare pînă la viteza nominală de citire de 10 ips.

Blocurile de informație din cadrul unui fișier sînt compuse din preambul, 256 de octeți de informație de orice tip și doi octeți de control (CRC), preambulul nefiind inclus în calculul codului de control. Ele sînt separate prin spații interbloc cu totul asemănătoare cu cel situat după blocul de identificare al fișierului. Ultimul bloc de informație al fișierului va fi mai deosebit, în sensul că la epuizarea informației utile, rutinele de I/E ale unităților de casete magnetice adaugă automat numărul necesar de octeți nuli pentru completarea celor 256 de octeți ai săi. De asemenea, spațiul neînregistrat, ce urmează ultimului bloc al fișierului, este mult mai mare decît un spațiu interbloc normal, el purtînd numele de spațiu interfișier (*file gap*). Detectarea sa permite interfeței decelarea sfîrșitului de fișier, respectiv începutului unui nou fișier, în cadrul operației de căutare a unui fișier, în care se efectuează salturi rapide peste un întreg fișier. Lungimea sa este astfel aleasă ca să permită: decelarea unui spațiu interbloc anormal de lung, atrăgînd după sine concluzia că este vorba de un spațiu interfișier, frînarea pînă la 0 de la viteza rapidă de căutare și repornirea benzii pînă la viteza nominală de citire de 10 ips necesară pentru citirea blocului de identificare al fișierului următor.

După cum a reieșit și din prezentarea de pînă acum, interfața poate cere derulorului să funcționeze cu două viteze diferite: viteza de 10 ips este viteza normală de scriere/citire a informației, iar viteza de 40 ips este viteza de căutare rapidă, la care se efectuează numai detecția spațiului interfișier.

6.8.2. DESCRIEREA UNITĂȚII DE CASETE MAGNETICE FOLOSITE

Unitatea de casete magnetice tip MFE 250B are următoarele caracteristici tehnice:

— posedă un cap magnetic permițînd citirea, scrierea și citirea imediat după scriere pe pista de date și, respectiv, citirea informației de comandă a mișcării pe pista de sincronizare;

— posedă un circuit de comandă a tensionării benzii magnetice, care asigură ca banda să fie în contact permanent cu capul, indiferent de variațiile de sarcină și, respectiv, de viteză, ale motoarelor;

— posedă un circuit de detecție optică a porțiunii transparente de început a casetei;

— permite detectarea încărcării casetei;

— permite detectarea stării de protejare a casetelor împotriva scrierii accidentale;

— viteza de derulare este comandată din exterior în plaja 2—120 ips; viteza nominală de scriere/citire este 10 ips, iar viteza de derulare rapidă în timpul căutării a fost aleasă 40 ips;

— temperatura de lucru:

— pentru derulor: 0—55°C;
— pentru casete: 10—45°C.

— variația garantată a vitezei benzii:

— ± 1% pe termen scurt;
— ± 3% pe termen lung.

— timpul maxim de start de la 0 la 10 ips: 50 ms (proporțional pentru viteze mai mari);

— timpul maxim de stop de la 10 ips la 0: 30 ms (proporțional pentru viteze mai mari);

— probabilitatea erorilor:

— 1 bit din 10^7 pentru erori „soft";
— 1 bit din 10^8 pentru erori „hard";

— tensiunile de alimentare folosite:

+5 V/maximum 1 A;
—5 V/maximum 0,5 A.

Semnalele de interfață ale derulorului se pot grupa în patru categorii:

— semnale de stare;
— semnale de comandă;
— semnale de date;
— alimentări.

Sensul de circulație al semnalelor este considerat în raport cu derulorul.

Semnalele de stare sînt ieșiri TTL care au următoarele funcțiuni:

— CASSETTE LOADED constituie un semnal activ jos, indicînd, atunci cînd este „0", că în unitate a fost inserată o casetă.

— END OF TAPE este un semnal care trece la „0" atunci cînd în derulor se află inserată o casetă, s-a dat unității o comandă de mișcare și sînt detectate în mod normal impulsurile de pe pista de sincronizare. Linia EOT trece la „1", indicînd o funcționare eronată, dacă: nu se detectează impulsuri de sincronizare într-un interval de 50 ms după inițierea unei comenzi de mișcare, sau dacă în timpul mișcării aceste impulsuri sînt absente pentru o durată mai lungă de 15 ms. Cauzele unei asemenea erori de funcționare pot fi: absența

depunerii de strat magnetic pe bandă, sfîrșitul fizic al benzii, bandă ruptă sau bandă blocată. După apariția unei condiții de EOT, derulorul nu va mai accepta nici o altă comandă de mișcare pînă la trimiterea unei comenzi de ștergere generală a derulorului, RESET.

— BUSY este un semnal care trece la „1" după recepționarea de către derulor a unei comenzi de mișcare. El rămîne „1" pînă la terminarea mișcării sau pînă la apariția unei condiții de EOT.

— WRITE PERMIT devine „0" cînd caseta montată în unitate nu este protejată împotriva scrierii accidentale, lucru care se obține prin înlăturarea limbii de plastic existente pe latura casetei opusă celei în care intră capul magnetic. În cazul unei casete protejate la scriere, linia WRITE PERMIT trece la „1" și curentul prin capul de scriere este inhibat.

— OPTICAL EOT/BOT trece la „0" atunci cînd porțiunea de bandă aflată în fața capului de scriere/citire este transparentă din punct de vedere optic. În cazul casetelor folosite, acest lucru are loc în două situații: pe porțiunea transparentă de material nemagnetic aflată la începutul și sfîrșitul benzii și la trecerea prin fața capului a celor două găuri de marcaj, aflate la ambele capete ale benzii pe porțiunea magnetică, la aproximativ 18 inches de porțiunea transparentă. Zona magnetică utilă a benzii este cea cuprinsă între cele două găuri amintite. Semnalul OPTICAL EOT/BOT este „1" cînd banda este netransparentă.

Semnalele de comandă sînt, de asemenea, în niveluri TTL, avînd următoarele funcțiuni:

— TAPE SPEED CONTROL CLOCK este un semnal de ieșire, care se prezintă sub forma unui tren de impulsuri rezultat din citirea și formarea semnalului de ceas preînregistrat pe pista de sincronizare, cu o spațiere echidistantă de 1 600 tranziții/inch.

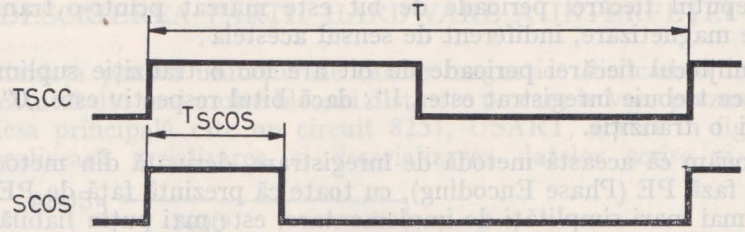
— SPEED CONTROL ONE-SHOT este un semnal de intrare pe care derulorul îl folosește pentru reglarea vitezei de transport a benzii. El trebuie furnizat din exterior cu ajutorul unui monostabil compensat la variațiile termice, de tip 74121. Acest semnal intervine direct în bucla de reglare a vitezei constante a benzii, în sensul că sistemul automat lucrează în așa fel încît să mențină factorul de umplere al acestui semnal la 24%.

Impulsurile TSCC (TAPE SPEED CONTROL CLOCK) și SCOS (SPEED CONTROL ONE-SHOT) sînt prezentate în figura 6.27, în care apar, de asemenea, și duratele semnalului SCOS necesare pentru funcționarea la diferite viteze.

— RESET este un semnal de intrare activ jos, care trebuie să se prezinte ca un impuls cu o durată minimă de 1 μs. O comandă RESET pregătește derulorul de lucru; de asemenea, această comandă este necesară după apariția semnalului EOT, pentru a putea continua lucrul.

— REVERSE GO este un semnal de intrare a cărui punere la „1" produce mișcarea înapoi a benzii cu viteza determinată de durata semnalului SCOS. Mișcarea încetează atunci cînd acest semnal redevine „0" sau la apariția lui EOT.

— FORWARD GO constituie un semnal de intrare similar cu precedentul, cu deosebirea că sensul mișcării este înainte. Ca regulă generală, în cazul schimbării sensului de mișcare sau al vitezei de derulare, comanda de mișcare



Viteza de derulare inches/s	Perioada ceasului preînregistrat T (μs)	Perioada semnalului SCOS T_{SCOS}(μs)
5	250	60
10	125	30
20	62,5	15
40	31,25	7,5
80	15,63	3,75

Fig. 6.27. Semnalele TSCC și SCOS și relația dintre durata lui SCOS și viteza de derulare a benzii

în sens contrar sau cu viteză diferită nu se dă decît după oprirea completă a mișcării benzii, ca efect al comenzii anterioare.

Semnalele de date sînt exprimate în niveluri TTL și au următoarele funcțiuni:

— DATA(+) este un semnal de ieșire care în mod obișnuit se găsește la „0", înregistrînd un impuls la „1" cu durata de 1 μs la fiecare tranziție pozitivă a fluxului magnetic, sesizată de capul de citire.

— DATA(—) este similar cu DATA(+), impulsul de 1 μs apărînd în acest caz pe tranziția negativă a fluxului magnetic pe bandă.

— DATA este un alt semnal de ieșire egal cu suma logică a precedentelor; pe această linie va apărea un impuls de 1 μs la fiecare tranziție a fluxului magnetic.

— FLUX0 și FLUX1 sînt semnale de intrare care comandă curentul prin capul de scriere. Dacă ambele linii sînt la „0" sau la „1", prin capul de scriere nu trece curent, dacă FLUX0 = „0" și FLUX1 = „1" se produce magnetizarea benzii în direcția Nord, în timp ce inversarea polarităților acestor semnale produce magnetizarea benzii în direcția Sud. Polarizarea normală a benzii în starea ștearsă este în direcția Nord.

Metoda de înregistrare folosită (Bi-Phase-Mark) se supune următoarelor reguli:

— fiecărui bit de date i se alocă o perioadă fixă, care se poate deduce din viteza de derulare și din densitatea de înregistrare, adică: 800 biți/inch × × 10 inches/s = 8 000 biți/s, corespunzînd unei perioade de 125 μs/bit;

— începutul fiecărei perioade de bit este marcat printr-o tranziție a fluxului de magnetizare, indiferent de sensul acesteia;

— la mijlocul fiecărei perioade de bit are loc o tranziție suplimentară dacă bitul ce trebuie înregistrat este „1"; dacă bitul respectiv este „0", nu se înscrie nici o tranziție.

Menționăm că această metodă de înregistrare, derivată din metoda modulației de fază PE (Phase Encoding), cu toate că prezintă față de PE avantajul unei mai mari simplități de implementare, este mai puțin fiabilă și mai sensibilă la zgomote decît PE. Totuși, la densitatea și viteza de înregistrare folosite, această metodă dă completă satisfacție.

În figura 6.28 este prezentată diagrama de timp a semnalelor de scriere și citire a informației, pentru secvența de biți 1011. Pe această figură se poate constata modul de folosire a semnalelor de scriere FLUX1 și FLUX0, ca și a semnalelor de citire DATA, DATA(+) și DATA(—).

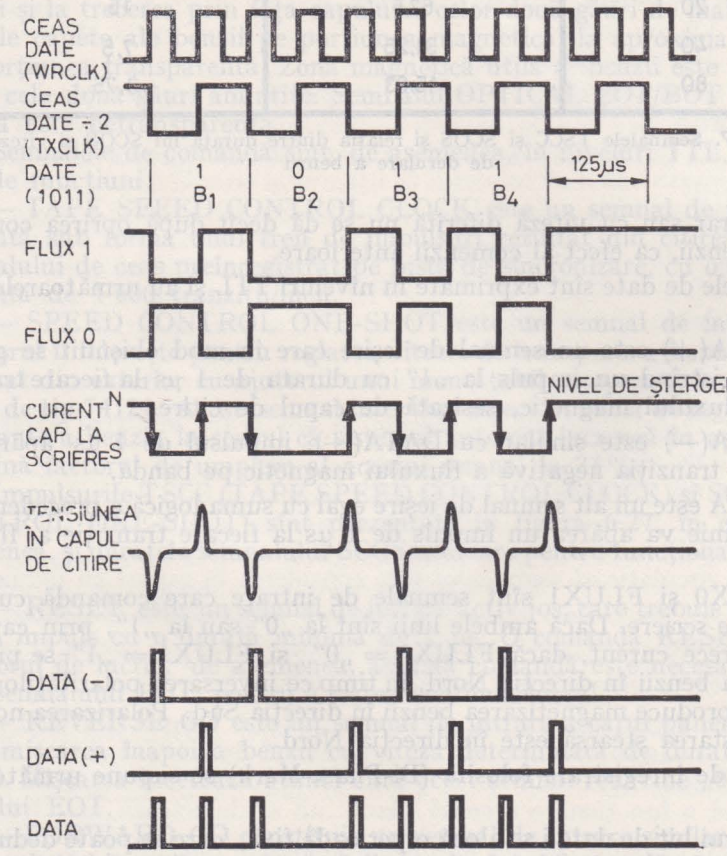
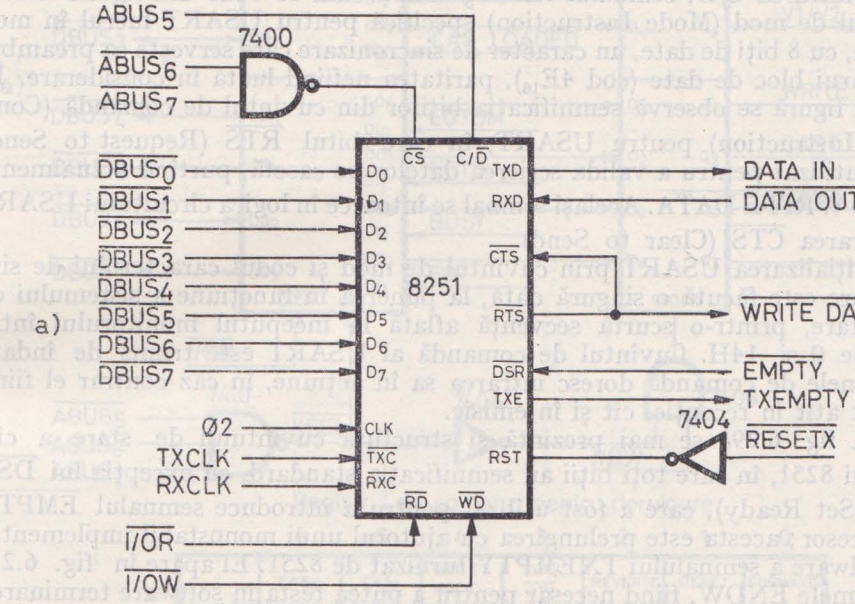


Fig. 6.28. Diagrama de timp a semnalelor de date în cazul metodei Bi-Phase-Mark

6.8.3. DESCRIEREA PĂRȚII HARDWARE A INTERFEȚEI

Interfața pentru unitățile de casete magnetice din cadrul lui SD-8080 este alcătuită din mai multe blocuri distincte pe care le vom prezenta separat.

Piesa principală este un circuit 8251, USART, arătat în figura 6.29*a*, care realizează serializarea și deserializarea datelor scrise și, respectiv,



Cuvîntul de mod 8251

7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0
1 CARACTER SYNC INTERN		FĂRĂ PARITATE		8 BIȚI		SINCRON	

Cuvîntul de comandă 8251

b)

7	5	5	4	3	2	1	0
ENABLE HUNT	RESET	WRITE DATA (RTS)	ERROR RESET	0	RX ENABLE	0	TX ENABLE

Cuvîntul de stare 8251

7	6	5	4	3	2	1	0
ENDW (DSR)	SYNC DETECT	x	OVERRUN ERROR	x	TX EMPTY	RX READY	TX READY

Fig. 6.29. Circuitul 8251-USART al interfeței pentru unitățile de casete magnetice (a, b)

citite pe/de pe unitatea de casete în lucru, și conectarea interfeței la magistrala de date a sistemului. Acest circuit este identificat de procesor prin adresele: de intrare, $9F_{16}$ pentru date și BF_{16} pentru cuvîntul de stare, și de ieșire, $9F_{16}$ pentru date și BF_{16} pentru comenzi (vezi tabelele 4.3 și 4.4), primind pentru aceasta pe intrarea de selecție $\overline{CS}$ semnalul $\overline{ABUS_7} \cdot \overline{ABUS_6}$, iar pe intrarea C/$\overline{D}$, semnalul $ABUS_5$. După cum se vede din figura 6.29*b*, cuvîntul de mod (Mode Instruction) specifică pentru USART lucrul în mod sincron, cu 8 biți de date, un caracter de sincronizare care servește ca preambul al fiecărui bloc de date (cod $4E_{16}$), paritatea nefiind luată în considerare. În aceeași figură se observă semnificația biților din cuvîntul de comandă (Command Instruction) pentru USART, în care bitul $\overline{RTS}$ (Request to Send) a fost utilizat pentru a valida scrierea datelor pe casetă, purtînd actualmente numele $\overline{WRITE\ DATA}$. Același semnal se întoarce în logica circuitului USART pe intrarea $\overline{CTS}$ (Clear to Send).

Inițializarea USART prin cuvîntul de mod și codul caracterului de sincronizare este făcută o singură dată, la punerea în funcțiune a sistemului de dezvoltare, printr-o scurtă secvență aflată la începutul monitorului între adresele 0 și 14H. Cuvîntul de comandă al USART este trimis de îndată ce rutinele de comandă doresc intrarea sa în acțiune, în caz contrar el fiind inhibat atît în recepție, cît și în emisie.

În fig. 6.29*b* se mai prezintă și structura cuvîntului de stare a circuitului 8251, în care toți biții au semnificația standard, cu excepția lui $\overline{DSR}$ (Data Set Ready), care a fost utilizat pentru a introduce semnalul $\overline{EMPTY}$ în procesor; acesta este prelungirea cu ajutorul unui monostabil implementat în hardware a semnalului TXEMPTY furnizat de 8251. El apare în fig. 6.29*b* sub numele ENDW, fiind necesar pentru a putea testa în software terminarea scrierii, pe timpul operației de citire de control după scriere.

Remarcăm că toți octeții de comandă sau stare prin care procesorul comunică cu acest circuit trebuie complementați, datorită negării liniilor magistralei de date.

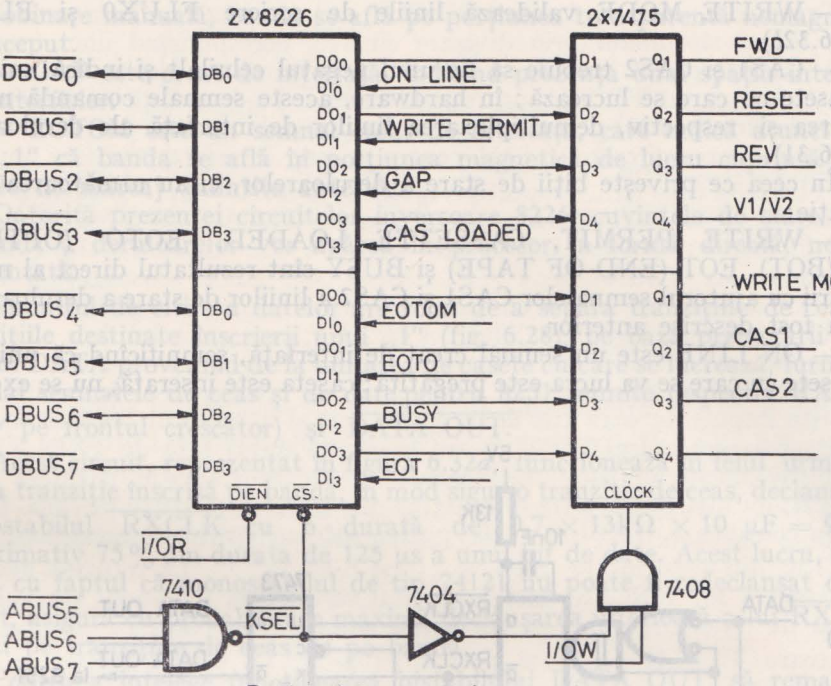
Fac excepție de la această regulă octeții de date citiți sau scrișl la 8251, pentru care s-au luat precauții speciale de negare a biților în logica de intrare, respectiv ieșire, din USART.

O altă cale de comunicație a procesorului cu această interfață este registrul de stări și de comenzi al deruloarelor, identificat în cadrul sistemului de I/E cu ajutorul adresei DF_{16} (tabelele 4.3 și 4.4). Schema de principiu a acestui registru este dată în figura 6.30. Biții care apar în cadrul registrului de comenzi pentru deruloare au următoarele semnificații:

— FORWARD comandă mișcarea înainte a unității de casete specificată de unul din biții CAS1/CAS2; pe baza lui se creează semnalele de comandă a deruloarelor 1FORWARD GO și 2FORWARD GO (fig. 6.31).

— RESET și REVERSE formează semnalele corespunzătoare de comandă a deruloarelor, pe baza modelului din figura 6.31.

— VIT 1/$\overline{VIT\ 2}$ este un semnal exploatat de interfață, ce indică atunci cînd este „1" că mișcarea se execută cu viteza normală de 10 ips, iar cînd e „0" se execută o mișcare rapidă cu viteza de 40 ips.



a)

Fig. 6.30. Schema de principiu și semnificația biților registrului de stări și de comenzi al interfeței pentru unitățile de casete magnetice (a, b)

Registrul de comenzi pentru deruloare

7	6	5	4	3	2	1	0
X	CAS_2	CAS_1	WRITE MODE	VIT1/$\overline{VIT2}$	REVERSE	RESET	FORWARD

b)

Registrul de stare a deruloarelor

7	6	5	4	3	2	1	0
EOT	BUSY	EOTO	EOTOM	CASS LOADED	GAP	WRITE PERMIT	ON LINE

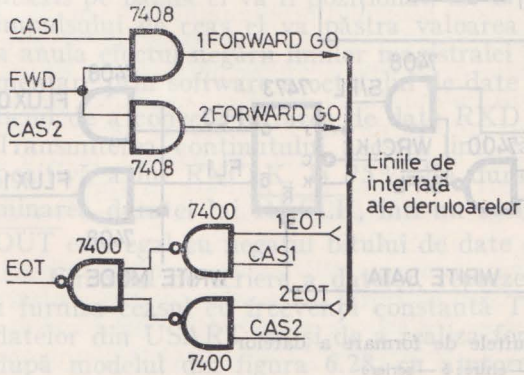


Fig. 6.31. Exemplu de multiplexare și demultiplexare a comenzilor și stărilor pe liniile de interfață ale deruloarelor

— WRITE MODE validează liniile de scriere FLUX0 și FLUX1 (fig. 6.32*b*).

— CAS1 și CAS2 trebuie să fie unul negatul celuilalt și indică unitatea de casete cu care se lucrează; în hardware, aceste semnale comandă multiplexarea și, respectiv, demultiplexarea liniilor de interfață ale deruloarelor (fig. 6.31).

În ceea ce privește biții de stare a deruloarelor, ei au următoarea semnificație:

— WRITE PERMIT, CASSETTE LOADED, EOTO (OPTICAL EOT/BOT), EOT (END OF TAPE) și BUSY sînt rezultatul direct al multiplexării cu ajutorul semnalelor CAS1 și CAS2 a liniilor de stare a deruloarelor, ce au fost descrise anterior.

— ON LINE este un semnal creat de interfață, semnificînd că unitatea de casete cu care se va lucra este pregătită: caseta este inserată, nu se execută

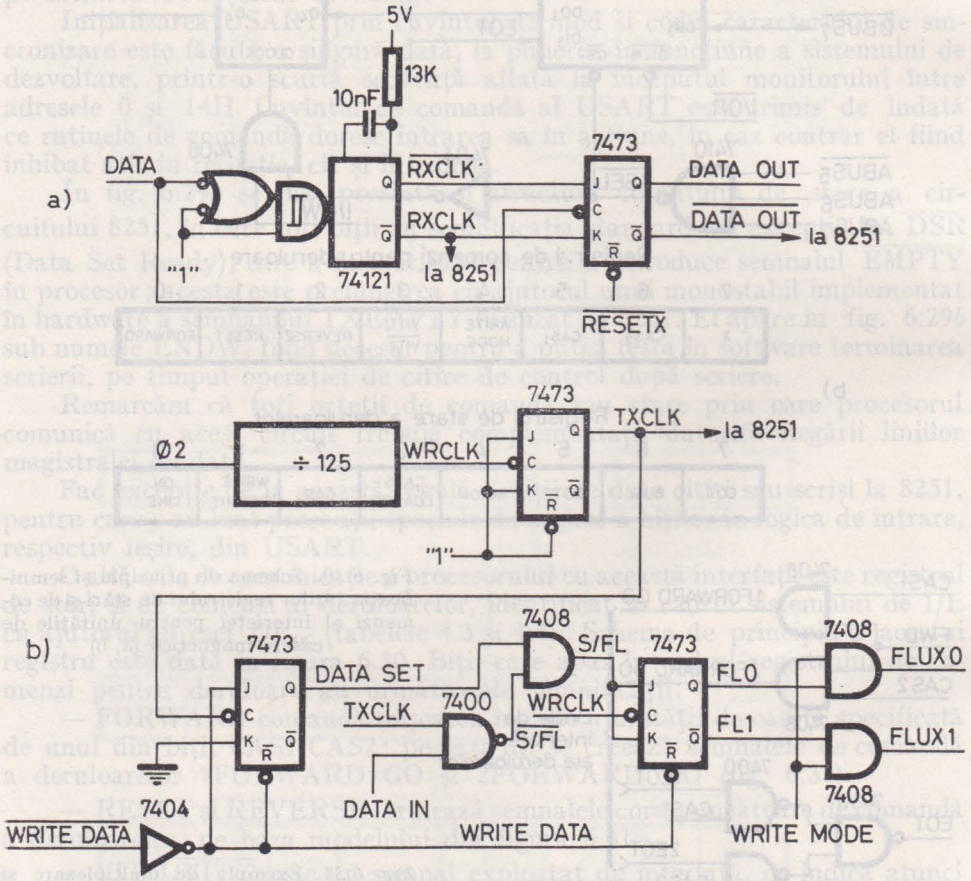


Fig. 6.32. Circuitele de formare a datelor:
a — citire; *b* — scriere

o rebobinare manuală, banda se află pe porțiunea transparentă nemagnetică de început.

— GAP este creat de interfață, indicînd prezența unui spațiu interbloc sau interfișier.

— EOTOM este un semnal creat de interfață, care indică atunci cînd este „1" că banda se află în porțiunea magnetică de lucru cuprinsă între găurile de marcaj amintite anterior.

Datorită prezenței circuitelor inversoare 8226, cuvintele de stare și de comandă a deruloarelor vor apărea în procesor în formă directă, necomplementată.

Circuitul de citire a datelor are rolul de a separa tranzițiile de ceas de tranzițiile destinate înscrierii unui „1" (fig. 6.28), pe baza prelucrării semnalului DATA provenind de la unitatea de casete cu care se lucrează, furnizînd în final semnalele de ceas și de date pentru 8251, numite respectiv RXCLK (activ pe frontul crescător) și $\overline{\text{DATA OUT}}$.

Acest circuit, reprezentat în figura 6.32*a*, funcționează în felul următor: prima tranziție înscrisă pe bandă, în mod sigur o tranziție de ceas, declanșează monostabilul $\overline{\text{RXCLK}}$ cu o durată de $0{,}7 \times 13\text{k}\Omega \times 10\ \mu\text{F} = 91\ \mu\text{s}$, aproximativ 75% din durata de 125 μs a unui bit de date. Acest lucru, coroborat cu faptul că monostabilul de tip 74121 nu poate fi redeclanșat odată pornit, asigură cu probabilitate maximă declanșarea ulterioară a lui $\overline{\text{RXCLK}}$ numai pe tranzițiile de ceas de pe bandă.

Pentru a înțelege funcționarea bistabilului DATA OUT, să remarcăm că fiecare tranziție de ceas pe linia DATA produce ștergerea lui, deoarece la începutul perioadei de bit $\overline{\text{RXCLK}}$ nu a fost încă declanșat, ceea ce atrage $J_{\text{DATA OUT}} = \overline{K}_{\text{DATA OUT}} =$ „0".

La aproximativ 62,5 μs de la declanșarea lui RXCLK, la mijlocul perioadei de bit, în cazul unui bit de date egal cu „1", pe bandă este înscrisă o tranziție suplimentară, care apare pe linia DATA sub forma unui impuls scurt de 1 μs, transmițîndu-se ca ceas al bistabilului DATA OUT. Cum la mijlocul bitului de date acest bistabil are pe intrările de date J și K respectiv semnalele $\overline{\text{RXCLK}}$ = „1" și RXCLK = „0", rezultă că în cazul unui „1" înscris pe bandă el va fi poziționat, iar în cazul unui „0", din cauza absenței impulsului de ceas el va păstra valoarea anterioară, rămînînd „0". Pentru a anula efectul negării liniilor magistralei de date, care ar implica o complementare prin software a octetului de date citit de la 8251, s-a recurs la artificiul de a conecta intrarea de date RXD a lui 8251 la ieșirea $\overline{\text{DATA OUT}}$. Transmiterea conținutului acestei linii către USART se face la tranziția pozitivă a lui RXCLK, la 75% din durata perioadei de bit, adică la terminarea duratei lui $\overline{\text{RXCLK}}$, într-un moment cînd conținutul liniei $\overline{\text{DATA OUT}}$ este egal cu negatul bitului de date curent ce a fost citit de pe bandă.

Circuitul de scriere a datelor, reprezentat în figura 6.32*b*, are rolul de a furniza ceasul cu frecvență constantă TXCLK, necesar pentru extragerea datelor din USART, ca și de a realiza forma de undă a magnetizării benzii după modelul din figura 6.28, cu ajutorul semnalelor FLUX0 și FLUX1.

Ceasul de scriere a datelor, WRCLK, este realizat pe baza ceasului cu cuarț Ø2 al sistemului prin divizare cu 125, corespunzînd unei frecvențe de 16 KHz, respectiv unei durate de 62,5 μs. Prin divizarea cu 2 a acestuia se obține semnalul TXCLK, un impuls simetric cu frecvența de 8 KHz, respectiv durata de 125 μs, egală cu durata perioadei de bit.

La activarea circuitului 8251 pentru scriere, interfața primește semnalul $\overline{\text{WRITE DATA}}$ = „0", respectiv WRITE DATA = „1". Valoarea anterioară a lui WRITE DATA fiind „0", bistabilii DATA SET și FL0 sînt ambii șterși în momentul acționării pentru scriere a circuitului USART. Aceasta corespunde lui FLUX1 = „1", FLUX0 = „0", adică sensului normal de polarizare a benzii neînscrise cu date. La prima tranziție negativă a ceasului TXCLK se poziționează bistabilul DATA SET, care are rolul de a sincroniza serializorul cu circuitul de formare a datelor, din acest moment circuitul de scriere avînd funcționarea autorizată. În continuare, bistabilul FL0 primește impuls de ceas atît la începutul, cît și la mijlocul fiecărei perioade de bit, deoarece ceasul WRCLK are frecvență dublă față de TXCLK. Secvența evenimentelor este următoarea: la fiecare tranziție negativă a lui TXCLK, un nou bit de date este extras din USART și apare pe linia DATA IN; în continuare, TXCLK = „0" pentru întreaga perioadă următoare a lui WRCLK, din acest motiv $\overline{\text{S/FL}}$ = „1" și S/FL = „1". La tranziția negativă următoare a lui WRCLK, bistabilul FL0 va bascula indiferent de valoarea bitului de date, înscriindu-se pe bandă tranziția de ceas. Pe semiperioada următoare TXCLK = „1" și deci $\overline{\text{S/FL}}$ = $\overline{\text{DATA IN}}$, atrăgînd S/FL = = $\overline{\text{DATA IN}}$.

Se deduce că la tranziția negativă următoare a lui WRCLK, bistabilul FL0 va bascula dacă bitul curent de date este „0". Observăm că informația înscrisă pe bandă este negarea bit cu bit a celei ce se primește pe linia DATA IN, deci ținînd cont de negarea introdusă de transportul informației pe magistrala de date, rezultă că pe bandă se înscrie informația existentă în acumulator în momentul executării instrucțiunii ØUT 9FH. Remarcăm că circuitul de formare a datelor înscrie datele pe bandă cu o întîrziere de o semiperioadă față de extragerea lor din serializor, ciclul de funcționare reluîndu-se la extragerea unui nou bit din USART, moment care coincide cu înscrierea pe bandă a tranziției corespunzătoare bitului de date precedent.

Linia WRITE MODE validează, sub controlul programului, transmiterea ieșirilor lui FL0 la liniile de comandă a curentului prin capul de scriere.

Pentru a asigura lucrul unităților de casete la viteza normală de 10 ips interfața posedă monostabilul SCOS1 cu durata T_1 = 30 μs; există un singur circuit de acest tip pentru cele două unități de casete, deoarece nu se poate efectua scrierea sau citirea pe ambele unități deodată. În schimb, pentru a putea efectua rebobinări și schimbări de casetă la o unitate în timp ce se lucrează cu cealaltă, s-au prevăzut doi monostabili de control pentru viteza de 40 ips, cu o durată T_2 = 7,5 μs, numiți SCOS21 și SCOS22.

În figura 6.33 sînt reprezentați monostabilii de comandă a vitezei. Semnalul SCOS1 apare ca răspuns la impulsurile de pe pista de sincronizare a oricăreia dintre casete, TSCC1 sau TSCC2, circuitul 74121 fiind validat numai cînd se lucrează cu viteza normală (V1/$\overline{\text{V2}}$ = „1"). Semnalul SCOS21

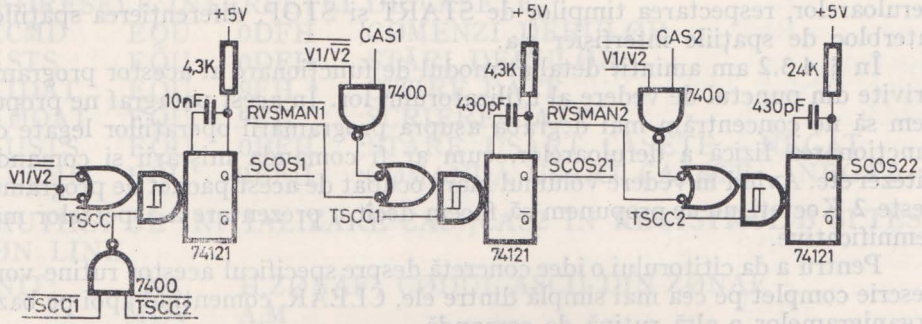


Fig. 6.33. Monostabilii de comandă a vitezei

apare ca răspuns la impulsul de pe pista de sincronizare TSCC1 a unității de casete numărul 1, dacă ne aflăm în rebobinare manuală ($\overline{RVSMAN1}$ = „0") sau dacă s-a comandat mișcarea unității de casete numărul 1 cu viteza rapidă ($V1/\overline{V2} \cdot CAS1$ = „1"). Semnalul SCOS22 este realizat la fel ca și SCOS21. Ecuațiile semnalelor SPEED CONTROL ONE-SHOT primite de unitățile de casete sînt:

$$1SCOS = SCOS1 + SCOS21, \text{ respectiv}$$
$$2SCOS = SCOS1 + SCOS22.$$

Semnalul GAP, care indică absența datelor pe caseta montată în unitatea activă în acel moment, este reuniunea ieșirilor negate a doi monostabili redeclanșabili de tip 74123, fiecare monostabil corespunzînd unei viteze de derulare. Monostabilul GAP1, activat cînd se lucrează la viteza normală de 10 ips, este declanșat de semnalul DATA, durata lui GAP1 fiind de 150 μs. În timpul derulării benzii la viteza normală, pe o porțiune de bandă înregistrată avem impulsuri pe linia DATA la un interval de cel mult 125 μs, GAP1 fiind continuu pe „1". La trecerea pe o porțiune neînregistrată, GAP1 nu mai primește semnal de declanșare și devine „0", acționînd GAP = „1". Monostabilul GAP2 funcționează după același principiu, sesizînd absența tranzițiilor pe pista de date în cazul derulării la viteza de 40 ips; durata sa este de 40 μs, perioada bitului fiind în acest caz de 31,25 μs.

Logica de panou furnizează pentru fiecare unitate de casetă semnalul ON LINE amintit anterior, indicînd că unitatea a fost pregătită de lucru de către operator. Starea fiecărei unități este monitorizată printr-o diodă electroluminiscentă plasată pe panou.

6.8.4. RUTINELE DE COMANDĂ

Rutinele de comandă sau *driver*-ii unităților de casete magnetice activează în mod corespunzător interfața hardware pentru a efectua structurarea datelor în blocuri și fișiere, realizarea controlului corectitudinii la citire și la scriere, comanda mișcării înainte sau înapoi a deruloarelor, comanda vitezei

deruloarelor, respectarea timpilor de START și STOP, diferențierea spațiilor interbloc de spațiile interfișier ș.a.

În § 4.3.2 am amintit detaliat modul de funcționare al acestor programe privite din punctul de vedere al utilizatorului lor. În acest paragraf ne propunem să ne concentrăm mai degrabă asupra programării operațiilor legate de funcționarea fizică a deruloarelor, cum ar fi comanda mișcării și comanda vitezei etc. Avînd în vedere volumul mare ocupat de acest pachet de programe, peste 2 Kocteți, nu ne propunem să facem decît o prezentare a aspectelor mai semnificative.

Pentru a da cititorului o idee concretă despre specificul acestor rutine vom descrie complet pe cea mai simplă dintre ele, CLEAR, comentînd apoi pe baza organigramelor o altă rutină de comandă.

```
;RUTINA DE ȘTERGERE CØMPLETĂ A CASETEI
;MØNTATE ÎN UNITATEA AL CĂREI NUMĂR E
;INDICAT DE CØDUL ASCII EXISTENT ÎN ZØNAF
CLEAR:  CALL   INIT    ;INIȚIALIZEAZĂ B,TEST ØN LINE
        CALL   LØAD    ;BANDA ÎN PØRȚIUNEA UTILĂ
        MVI    A,FØR+V1+WRMD
        ØRA    B       ;CØMANDĂ DE MIȘCARE LA VITEZA 1,
                       SCRIERE
        ØUT    KCMD
        IN     KCMD    ;ADU STARE DERULØR
        ANI    EØTM
        JNZ    $-4     ;AȘTEAPTĂ GAURA DE MARCAJ
        CALL   TM53    ;TEMPØRIZARE 53 MS
        CALL   STØP1   ;ØPRIRE DE LA VITEZA 1
        CALL   REW     ;REBØBINARE
        RET

;CØMENZI DERULØR
        FØR    EQU 1   ;FØRWARD
        RES    EQU 2   ;RESET
        REV    EQU 4   ;REVERSE
        V1     EQU 8   ;VITEZA 1 = 10 IPS
        V2     EQU 0   ;VITEZA 2 = 40 IPS
WRMD    EQU    10H     ;WRITE MØDE
CAS1    EQU    20H     ;CASETA 1
CAS2    EQU    40H     ;CASETA 2

;BIȚI DE STARE DERULØR
ØNL     EQU    1       ;ØN LINE
WRP     EQU    2       ;WRITE PERMIT
GAP     EQU    4       ;SPAȚIU INTERBLØC
CSL     EQU    8       ;CASSETTE LØADED
EØTM    EQU    10H     ;EØTØM
EØTØ    EQU    20H     ;EØT ØPTIC
BSY     EQU    40H     ;BUSY
EØT     EQU    80H     ;END ØF TAPE
```

```
;ADRESELE INTERFEȚEI DE CASETE
KCMD    EQU   0DFH   ;CØMENZI DERULØR
KSTS    EQU   0DFH   ;STĂRI DERULØR
KIDAT   EQU   9FH    ;CITIRE DATE
KØDAT   EQU   9FH    ;SCRIERE DATE
KUSTS   EQU   0BFH   ;STARE USART CASETE — NEGAT
KUCMD   EQU   0BFH   ;CØMANDĂ USART CASETE — NEGAT

;RUTINA DE INIȚIALIZARE CAS1/CAS2 ÎN REGISTRUL B ȘI TEST
;ØN LINE
INIT:   LXI   H,ZØNAF; CØDUL ASCII DIN ZØNAF
        MØV   A,M
        CPI   31H    ;ESTE 1 ?
        JNZ   INIT2  ;NU
        MVI   A, CAS1 ;CAS1 ÎN A
        MØV   B,A    ;CAS1 ÎN B
        JMP   INIT2+3
INIT2:  MVI   A, CAS2  ;CAS2 ÎN A
        MØV   B, A     ;CAS2 ÎN B
        ØUT   KCMD     ;INIȚIALIZEAZĂ CAS1/CAS2 LA INTER-
                       ;FAȚĂ PENTRU A PUTEA TESTA
                       ;ULTERIØR STAREA UNITĂȚII DØRITE
        IN    KSTS     ;ADU STARE DERULØR
        ANI   ØNL      ;REȚINE ØN LINE
        JZ    $-4      ;AȘTEAPTĂ PÎNĂ ØN LINE
        RET

;RUTINA DE ADUCERE A BENZII ÎN PØRȚIUNEA MAGNETICĂ
;UTILĂ
LØAD:   MVI   A,RES    ;RESET DERULØR
        ØRA   B        ;ADAUGĂ CAS1/CAS2
        ØUT   KCMD
        DI             ;DEZACTIV. ÎNTRERUPERILE PENTRU
                       ;A NU PERTURBA TESTELE CE SE FAC
                       ;ÎN TIMPUL MIȘCĂRII
        MVI   A,FØR+V2
        ØRA   B        ;REPEDE ÎNAINTE
        ØUT   KCMD
LØAD1:  IN    KSTS     ;TEST EØT — DACĂ EØT BUCLEAZĂ
        ØRA   A        ;RESET + MIȘCARE PENTRU A
        JM    LØAD     ;SCØATE BANDA DE PE CAPĂTUL
                       ;TRANSPARENT
        ANI   BSY      ;TEST BUSY PENTRU A
        JZ    LØAD     ;CØNTRØLA DACĂ MIȘCARE
        IN    KSTS     ;BANDA ÎN MIȘCARE, TEST EØTØM
        ANI   EØTM
        JZ    $-4      ;BUCLEAZĂ PÎNĂ EØTØM
        CALL  STØP2    ;ØPRIRE DE LA VITEZA 2
```

```
        EI              ;ACTIVEAZĂ LA LØC ÎNTRERUPERILE
        RET
;RUTINA DE TEMPØRIZARE 53 MS
TM53:   LXI   H,5C8H    ;CØNSTANTA TEMPØRIZĂRII
        DCX   H         ;DECREMENTARE CØNTØR
        PUSH  D
        LXI   D,0       ;VALØAREA FINALĂ CØNTØR
        CALL  CØMP      ;CØMPARAȚIE
        PØP   D
        JNZ   TM53+3    ;REIA BUCLA
        RET
;RUTINA DE COMPARAȚIE (D,E) CU (H,L)
CØMP:   MØV   A,D
        CMP   H
        RNZ
        MØV   A,E
        CMP   L
        RET
;RUTINA DE ØPRIRE DE LA VITEZA 1=10 IPS
STØP1:  MVI   A,0       ;CØMANDA DE ØPRIRE
        ØRA   B
        ØUT   KCMD
        CALL  TM30      ;TEMPØRIZARE 30 MS
        RET
;RUTINA DE REBØBINARE PÎNĂ PE CAPĂTUL TRANSPARENT
REW:    MVI   A,REV+V2  ;REPEDE ÎNAPØI
        ØRA   B         ;ADAUGĂ NR. CASETĂ
        ØUT   KCMD
        IN    KSTS      ;AȘTEAPTĂ EØT
        ANI   EØT
        JZ    $-4
        MVI   A,RES     ;RESET
        ØRA   B
        ØUT   KCMD
        RET
```

S-au omis din prezentare subrutinele: TM30, analoagă cu TM53, dar realizînd o întîrziere de 30 ms, și STØP2, al cărei efect e oprirea mișcării benzii de la viteza de 40 ips la 0, timpul de STOP fiind în acest caz de 72 ms.

Se impun cîteva observații referitoare la tehnicile de programare folosite în cadrul exemplului prezentat. Fiecare operație e descompusă în acțiuni mai simple implementate prin subrutine elementare foarte apropiate de nivelul hardware al interfeței. Aceasta conferă modularitate programului; subrutinele elementare cum ar fi INIT, STOP1/STOP2, REW ș.a. fiind folosite în comun de toți *driver*-ii. Pentru a degreva interfața hardware, măsurarea

timpului de START sau STOP, ca și a lungimii spațiilor interbloc sau interfișier este făcută prin bucle de întîrziere programate, scrise sub forma unor subrutine, ca de exemplu TM30, TM53 ș.a. Folosirea unității centrale pentru măsurarea timpului impune dezactivarea sistemului de întreruperi pe durata intrării în acțiune a rutinelor de temporizare, pentru a nu falsifica rezultatul măsurătorii de timp. Întreruperile sînt de asemenea dezactivate pe timpul funcțiilor legate de mișcarea benzii într-o buclă programată controlată de un indicator de stare al unității de casete (vezi cazul subrutinei LOAD). Putem conchide spunînd că programarea la acest nivel solicită din partea programatorului atît o bună cunoaștere a posibilităților unității centrale, cît și o cunoaștere intimă a structurii și caracteristicilor hardware-ului de sub comanda programului.

Vom descrie acum la nivel de organigramă comentată una dintre rutinele de comandă ale interfeței pentru unitățile de casete magnetice: KI (Cassette Input).

După cum am amintit și la § 4.3.2, KI citește inițial un bloc de 256 octeți de pe caseta specificată în ZONAF, efectuează controlul de validitate al datelor și apoi furnizează programului principal datele caracter cu caracter la fiecare chemare ulterioară. Cînd toți octeții unui bloc au fost transmiși, se efectuează automat citirea unui nou bloc pentru a reîmprospăta conținutul bufferului de intrare. Rutina KI presupune că la primul apel capul de citire a fost anterior poziționat de către rutina de căutare fișier, LOOK, pe spațiul interbloc aflat între blocul de identificare și primul bloc de date al fișierului dorit.

Pentru înțelegerea organigramei rutinei KI, prezentate în figura 6.34, este necesar să precizăm care sînt și la ce servesc locațiile de memorie RAM, alocate rutinelor de comandă a unităților de casete magnetice.

Locațiile RAM folosite de aceste programe sînt situate între adresele $BEFH_{16}$ și $BFFF_{16}$ inclusiv (fig. 6.35*a*). În această zonă de memorie sînt plasați cei șapte octeți ce formează ZONAF (fig. 4.30), urmează o zonă de doi octeți al cărui conținut reprezintă adresa curentă în bufferul de intrare/ieșire al rutinelor, la adresa următoare se găsește un octet folosit drept „cuvînt de stare a programului" (CSP) și, în sfîrșit, urmează cei 258 de octeți ai bufferului, corespunzînd adresei simbolice BUFF.

Pentru comunicarea între *driver*-ii unităților de casete magnetice și între apelurile succesive ale aceluiași *driver* se folosește „cuvîntul de stare a programului"; biților acestuia li s-au atribuit semnificațiile prezentate în figura 6.35*b*.

După salvarea registrelor programului apelant și inițializarea hardware și software a lucrului cu unitatea specificată în ZONAF, rutina KI testează bitul 5 al CSP, pentru a vedea dacă nu cumva bufferul de intrare este gol. Cazul cînd există caractere în buffer este banal; caracterul curent este plasat în registrul C pentru a fi predat programului chemător, se actualizează pointerul în buffer, se pune „1" în bitul 5 al CSP dacă bufferul a fost epuizat, caracterul de predat programului chemător este mutat în A, se refac apoi registrele-utilizator și se revine în programul principal. Dacă bufferul de intrare este gol, se începe citirea unui nou bloc, compusă dintr-o succesiune de acțiuni elementare figurate în mod explicit pe organigramă în cadrul secvenței de program

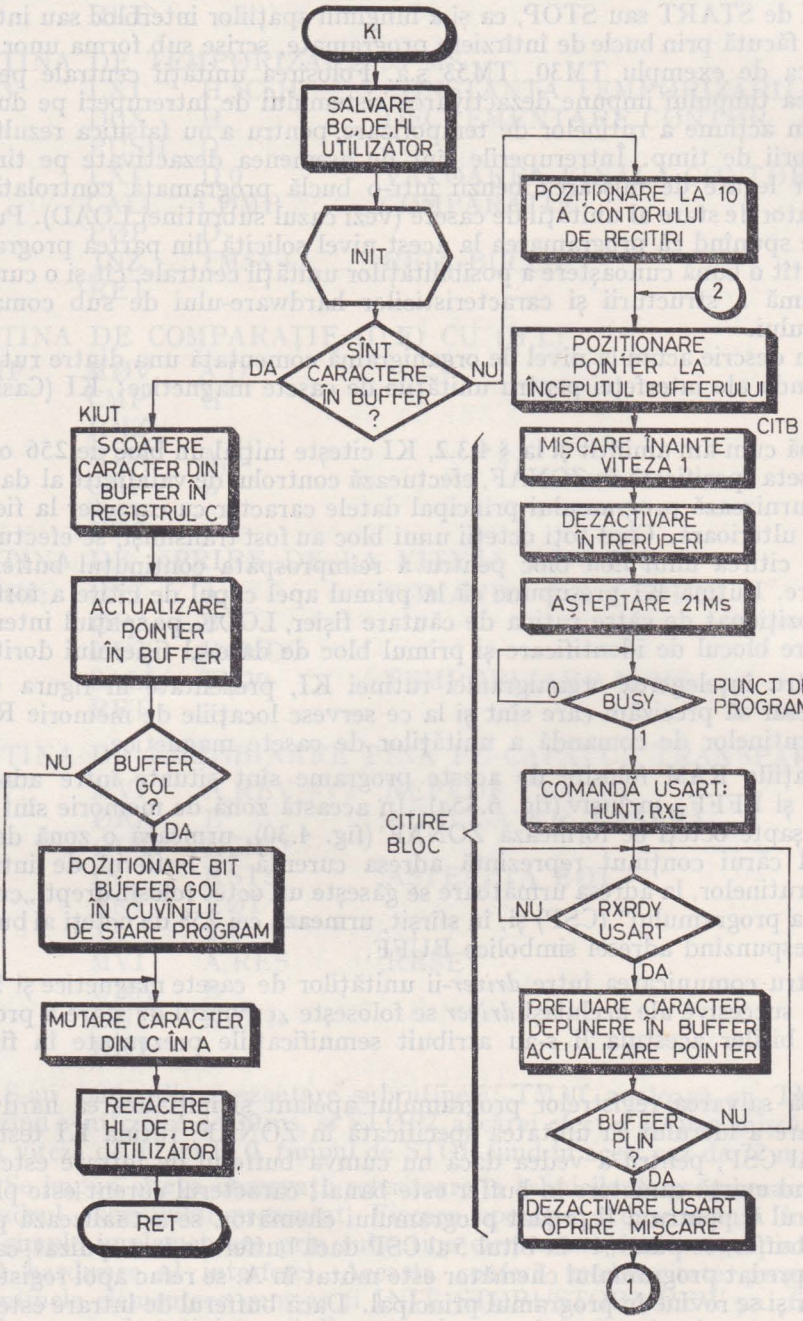
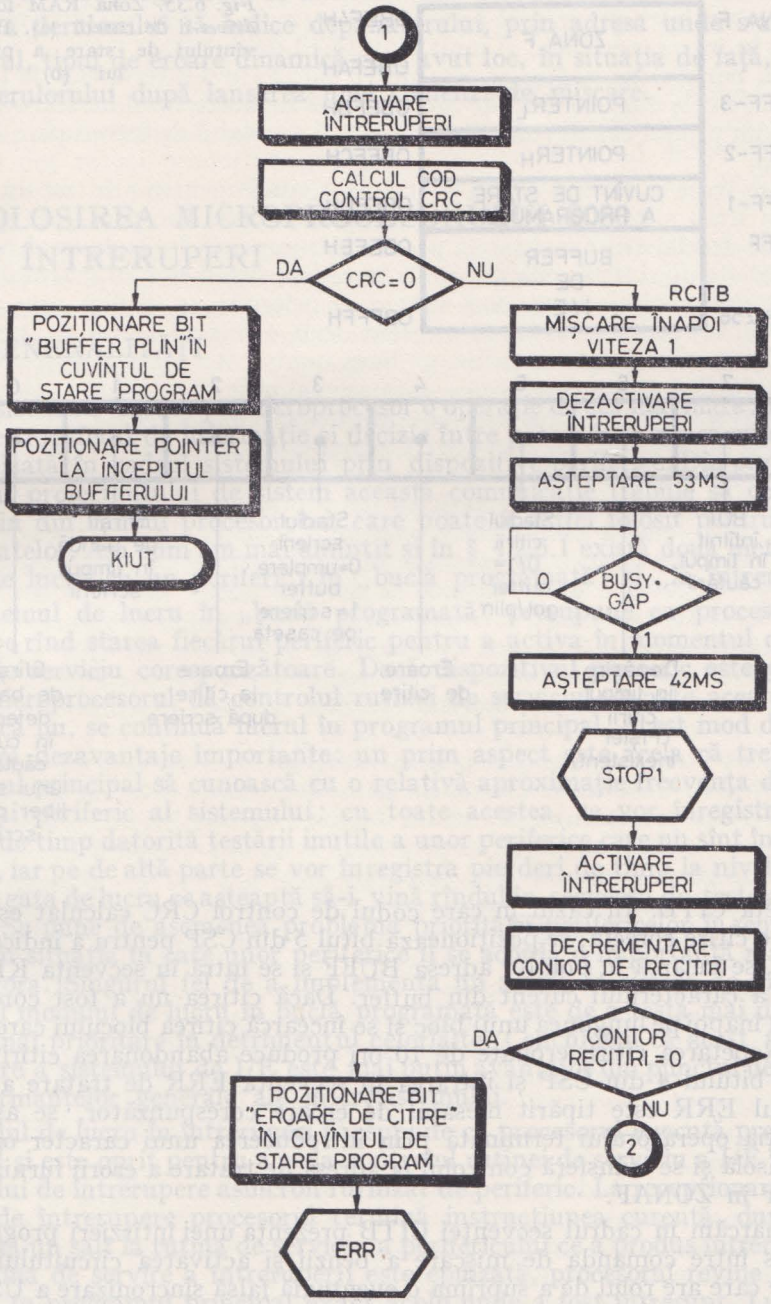


Fig. 6.34. Organigrama



rutinei KI (Cassette Input)

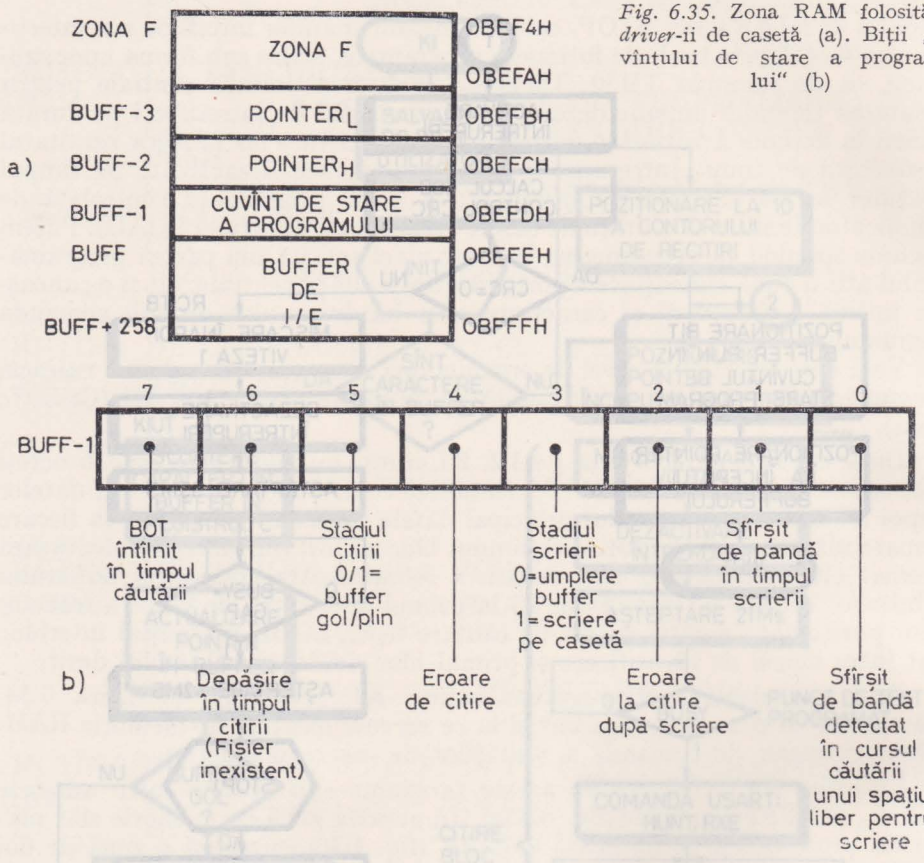


Fig. 6.35. Zona RAM folosită de *driver*-ii de casetă (a). Biții „cuvîntului de stare a programului" (b)

cu eticheta CITB. În cazul în care codul de control CRC calculat este nul indicînd o citire corectă, se poziționează bitul 5 din CSP pentru a indica „buffer plin", se încarcă în pointer adresa BUFF și se intră în secvența KIUT de scoatere a caracterului curent din buffer. Dacă citirea nu a fost corectă se dă banda înapoi pe lungimea unui bloc și se încearcă citirea blocului care a fost corect. Repetarea citirii eronate de 10 ori produce abandonarea citirii, poziționarea bitului 4 din CSP și intrarea în secvența ERR de tratare a erorii. În cadrul ERR este tipărit mesajul de eroare corespunzător, se așteaptă intervenția operatorului terminată prin introducerea unui caracter oarecare de la consolă și se transferă controlul la adresa de tratare a erorii furnizată de utilizator în ZONAF.

Remarcăm în cadrul secvenței CITB prezența unei întîrzieri programate de 21 ms între comanda de mișcare a benzii și activarea circuitului 8251, întîrziere care are rolul de a suprima o eventuală falsă sincronizare a USART-ului datorată variației fluxului magnetic prin capul de citire pe seama mișcării neuniforme a benzii ce are loc în timpul accelerării de la 0 la viteza nominală.

„Punctul de test programat" ce urmează are rolul ca în cazul unei funcționări eronate a derulorului să indice depanatorului, prin adresa unde s-a blocat procesorul, tipul de eroare dinamică ce a avut loc, în situația de față, neocuparea derulorului după lansarea unei comenzi de mișcare.

6.9. FOLOSIREA MICROPROCESORULUI 8080 ÎN ÎNTRERUPERI

6.9.1. GENERALITĂȚI

În sistemele bazate pe microprocesor o operație de cea mai mare însemnătate este transferul de informație și decizie între procesor și lumea exterioară, materializată în cadrul sistemului prin dispozitive periferice. Din punctul de vedere al proiectantului de sistem această comunicație trebuie să ocupe cît mai puțin din timpul procesorului, care poate fi astfel folosit pentru prelucrarea datelor. Așa cum am mai amintit și în § 4.2.5.1 există două metode generale de lucru cu un periferic: în „buclă programată" și „în întreruperi".

Sistemul de lucru în „buclă programată" presupune ca procesorul să testeze pe rînd starea fiecărui periferic pentru a activa în momentul oportun rutina de serviciu corespunzătoare. Dacă dispozitivul periferic este gata de lucru, microprocesorul dă controlul rutinei de serviciu asociate acestui periferic, dacă nu, se continuă lucrul în programul principal. Acest mod de lucru are cîteva dezavantaje importante: un prim aspect este acela că trebuie ca programul principal să cunoască cu o relativă aproximație frecvența de lucru a fiecărui periferic al sistemului; cu toate acestea, se vor înregistra mari pierderi de timp datorită testării inutile a unor periferice care nu sînt încă gata de lucru, iar pe de altă parte se vor înregistra pierderi de timp la nivelul unui periferic gata de lucru ce așteaptă să-i vină rîndul în secvența de teste a procesorului. Se pune de asemenea problema priorității perifericelor în sensul de a ne afla în situația în care unor periferice li se acordă o importanță mai mare decît altora. Singurul fel de a implementa un astfel de sistem de priorități în cadrul modului de lucru în buclă programată este de a testa mai des perifericele mai prioritare în detrimentul celorlalte. Conchidem că acest mod de organizare a sistemului de I/E este mai puțin avantajos din punctul de vedere al performanțelor generale ale microsistemului.

Modul de lucru în întreruperi presupune că procesorul execută programul principal și este oprit pentru a ceda controlul rutinei de serviciu a I/E pe baza semnalului de întrerupere asincron furnizat de periferic. La recepționarea semnalului de întrerupere procesorul termină instrucțiunea curentă, după care se execută un salt la rutina de serviciu a perifericului ce a produs întreruperea. Cînd rutina de servire a întreruperii este epuizată, procesorul revine în mod automat în programul principal exact acolo unde a fost întrerupt. Lucrul în întreruperi este evident mai avantajos din punctul de vedere al folosirii timpului

unității centrale, dar presupune tehnici de programare complicate pentru comunicarea între programul principal și rutinele de serviciu, sistemele funcționînd în întreruperi fiind și mult mai dificil de depanat.

Din punctul de vedere al organizării hardware a interfețelor de I/E, diferența între cele două moduri de lucru este că linia sau bitul de stare al perifericului din primul caz devine în al doilea caz semnalul de întrerupere al dispozitivului.

Vom trece în revistă cîteva concepte esențiale pentru arhitectura și funcționarea unui sistem de I/E lucrînd în întreruperi. Începem prin a aborda modul de decelare a cauzei care a produs întreruperea, legat de implementarea obligatorie a unui sistem de priorități în cadrul sistemelor cu mai multe cauze de întrerupere. Necesitatea unui sistem de priorități se explică prin aceea că procesorul nu poate servi la un moment decît o singură cauză de întrerupere din mulțimea celor active, aleasă pe baza unui sistem de prioritate. O primă metodă *(polling)* este producerea unui semnal unic de întrerupere ca reuniune a semnalelor de întrerupere asociate diferitelor cauze. Pe baza acestui semnal intră în acțiune rutina de serviciu a întreruperilor, care testează pe rînd fiecare periferic, pentru a afla pe cel care a produs întreruperea și a-i introduce în lucru rutina de I/E specifică. Ordinea de prioritate este în acest caz dată de ordinea de testare a perifericelor, primul dispozitiv de I/E interogat fiind evident cel mai prioritar. În contrast, în cazul metodei de vectorizare a întreruperilor *(vectoring)*, dintre semnalele de întrerupere active la un moment dat se selectează prin hardware numai cauza cea mai prioritară, furnizîndu-se procesorului codul întreruperii cu nivel de prioritate superior care poartă numele de „vector de întrerupere". Pe baza acestui vector, procesorul introduce în execuție rutina de serviciu specifică a cauzei ce a declanșat întreruperea.

Un alt mecanism esențial pentru funcționarea corectă a unui sistem de I/E lucrînd în întreruperi este acela de activare/dezactivare a sistemului de întreruperi al procesorului. Pentru a înțelege necesitatea acestui mecanism, vom spune că sistemul de întreruperi va fi, în general, dezactivat după punerea în funcțiune a procesorului prin inițializarea generală, permițînd astfel pregătirea rutinelor de serviciu a întreruperilor, cu alte cuvinte inițializarea software a sistemului de întreruperi. După executarea acestei sarcini procesorul va putea activa sistemul de întreruperi cu ajutorul unei instrucțiuni speciale, în cazul lui 8080 ea se numește EI (Enable Interrupts). Producerea și acceptarea unei întreruperi dezactivează automat sistemul de întreruperi pentru ca o altă cauză să nu poată întrerupe în mod intempestiv rutina de serviciu în curs. Folosind aceeași instrucțiune de activare a sistemului de întreruperi, rutina de serviciu îl va putea reintroduce în acțiune atunci cînd acest lucru nu mai poate produce defecțiuni în tratarea întreruperii anterior acceptate.

Anumite tipuri de microprocesoare (8085 sau Z80) posedă aparte întreruperi nemascabile care nu pot fi activate/dezactivate cu ajutorul instrucțiunilor amintite. Un gen de întrerupere nemascabilă este întreruperea semnalînd căderea tensiunii de alimentare.

Pentru a permite o mare flexibilitate în scrierea programelor de aplicație, sistemele de întreruperi evoluate permit mascarea/demascarea temporară

a unui nivel de întrerupere, înțelegînd prin mascare faptul că semnalul de întrerupere corespunzînd unui nivel mascat are calea de transmitere spre procesor temporar blocată.

Deosebirea mascării/demascării unui nivel față de activarea/dezactivarea sistemului de întreruperi este că în cazul mascării anumitor niveluri, cele nemascate pot totuși produce întreruperi, iar cînd sistemul de întreruperi e dezactivat nu se poate produce nici o întrerupere, blocîndu-se mecanismul de acceptare a lor de către procesor.

Există de asemenea un mecanism tipic de tratare hardware a semnalelor provenind de la sursele de întreruperi. Vom releva faptul că un asemenea semnal nu poate fi decît periodic, cu toate că frecvența lui de repetiție poate fi variabilă în limite largi; un semnal care stă continuu la „0" sau la „1" nu poate, desigur, indica în nici un fel momentul producerii unui eveniment extern procesorului. Rezultă că circuitul de tratare a semnalelor de întrerupere trebuie să fie sensibil la unul dintre fronturile semnalului de intrare provenind de la periferic; în acest fel se asigură independența funcționării circuitului de tratare de durata semnalului de întrerupere. Circuitul de intrare va fi deci în mod necesar un *latch* sensibil pe front (*edge-triggered*). Introducerea acestui latch de intrare atrage necesitatea existenței unei căi de ștergere a lui în vederea pregătirii lui pentru o acționare ulterioară. Procedeul se numește achitarea întreruperii, logica de ștergere a *latch*-ului de intrare aflîndu-se sub controlul programului, astfel că acest *latch* poate fi rearmat (șters) de rutina de serviciu, cînd tratarea acelei întreruperi a fost terminată. Ajunși la acest punct ne putem pune următoarea întrebare: dar dacă acest mecanism conduce la pierderea tranziției pe linia de intrare, însemnînd de fapt că unele dintre întreruperile ce survin pe același nivel sînt neglijate? Răspunsul este simplu: din start se pornește de la ideea că perioada de repetiție a fenomenului extern este suficient de mare pentru a permite încadrarea intervalului de timp necesar pentru executarea rutinei de serviciu. În caz contrar sistemul de calcul nu are viteza necesară pentru a funcționa în aplicația dată.

În figura 6.36*a* și *b* sînt prezentate schemele de principiu ale unui sistem de întreruperi cu 8 niveluri pentru microprocesorul 8080 cu priorități hardware *(vectoring)* și, respectiv, software *(polling)*. Observăm pe ambele figuri prezența celor 8 *latch*-uri de intrare și a registrului biților de măști care este programat ca o ieșire a microprocesorului. De asemenea, în ambele cazuri, linia de întrerupere INT de intrare în procesor este suma logică a ieșirilor reprezentînd produsele logice ale ieșirilor cîte unui *latch* de intrare cu bitul de mască asociat. În schema cu priorități hardware există o logică specializată a cărei piesă principală este un codor de prioritate cu 8 intrări care generează vectorul de întrerupere, instrucțiunea de achitare trimisă de procesor referindu-se în mod implicit la *latch*-ul celui mai înalt nivel de întrerupere activ în acel moment. Pe schema cealaltă este necesară prezența unui registru de 8 biți al întreruperilor active, care poate fi citit de procesor și, de asemenea, un registru de 8 biți de ieșire, prin care procesorul poate achita întreruperea pe care a găsit-o ca fiind cea mai prioritară. În legătură cu schema din figura 6.36*a*) putem face observația că logica de întrerupere poate fi implementată în mod distribuit, fiecare periferic generînd propriul său vector de întrerupere,

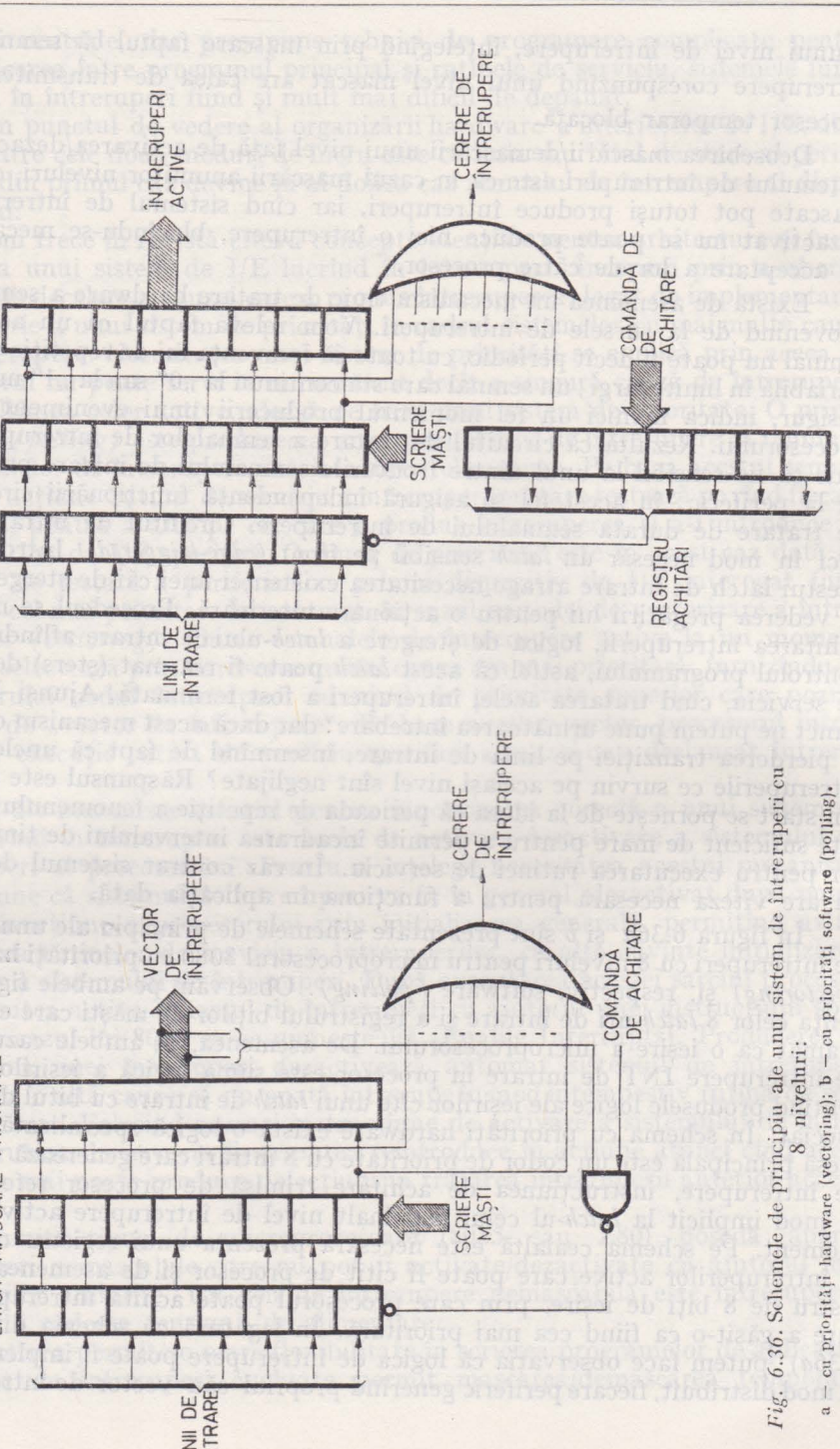


Fig. 6.36. Schemele de principiu ale unui sistem de întreruperi cu 8 nivelurî:

a — cu priorităţi hardware (vectoring); b — cu priorităţi software (polling).

prioritatea fiind materializată prin ordinea de cablare a perifericelor într-un lanț de priorități *(daisy chain)*.

În ceea ce privește tehnica programării sistemelor funcționînd în întreruperi, problema are două aspecte importante: scrierea rutinelor de serviciu a întreruperilor și metodele de comunicație și de sincronizare între rutinele de serviciu și programul principal. În funcție de diferite criterii proiectantul de sistem poate aloca sarcini mai complexe rutinelor de întreruperi sau poate scrie rutine de serviciu foarte simple plasînd capacitatea de decizie exclusiv la nivelul programului principal. Această opțiune influențează în mod hotărîtor modul de sincronizare dintre rutinele de serviciu și programul principal.

O anumită rutină de serviciu intră în acțiune la apariția cauzei externe ce îi este asociată. Operațiile ce le execută sînt de regulă următoarele:

— salvarea contextului procesorului: registre, numărător de adrese etc.;
— rezolvarea cererii externe printr-o acțiune specifică: introducerea unui caracter, preluarea unui octet de date, transmiterea unei comenzi către un dispozitiv extern etc.;
— activarea modului de întrerupere în chestiune pentru a-l pregăti în vederea unei folosiri ulterioare;
— restaurarea contextului procesorului;
— activarea sistemului de întreruperi;
— revenirea în programul principal la instrucțiunea curentă.

Programul principal comunică cu rutinele de serviciu prin intermediul unor *buffere* de date ce sînt exploatate în comun, umplute de programul principal și golite de rutina de serviciu în cazul unei rutine de ieșire de date din calculator și respectiv invers pentru o rutină de intrare de date, a unor pointeri, indicatori de stare etc.

O soluție de implementare este ca acțiunea ce se execută la intrarea în lucru a unei rutine de serviciu să fie variabilă și să poată fi schimbată fie de programul principal, fie de rutina însăși, în funcție de stadiul de execuție al algoritmului.

O rutină de serviciu poate în acest caz funcționa după modelul unui automat secvențial a cărui stare următoare depinde de starea actuală și de anumite variabile de intrare, avînd generat în fiecare stare un set de ieșiri determinate, în acest caz acțiuni specifice.

Mecanismul de întreruperi al microprocesorului 8080 folosește o linie de întrerupere INT, linia de achitare $\overline{\text{INTA}}$, bistabilul INTE de activare/dezactivare a sistemului de întreruperi, fiind admise 8 niveluri de întrerupere vectorizate cu ajutorul instrucțiunilor speciale RST n, $n = 0, 1, ..., 7$, care realizează automat saltul la adresele de tratare 0, 8, 16 ... de la începutul memoriei. Funcționarea acestui mecanism a fost descrisă în detaliu în paragraful 4.2.4. Reamintim că instrucțiunea RST n, o instrucțiune de tip CALL pe un singur cuvînt, ce produce saltul la adresa $8n$, este forțată pe magistrala de date a sistemului de o logică specializată în timpul ciclului ÎNTRERUPERE.

Avantajele acestui tip de instrucțiune sînt: ocupă un singur octet și durează un singur ciclu de execuție; permite folosirea a 8 vectori de întrerupere diferiți; adresele de tratare sînt suficient de distanțate (8 octeți) pentru a permite introducerea unor instrucțiuni JMP, de salt, la începutul adevăra-

telor rutine de tratare care nu au loc suficient în 8 octeți; în sfîrșit, hardware-ul exterior procesorului este destul de ușor de implementat, deoarece codul instrucțiunii are cinci unități, ceilalți trei biți putînd fi preluați de la ieșirea unui codor de prioritate cu 8 intrări. Instrucțiunea RST prezintă și dezavantaje: numărul de vectori de întrerupere este limitat; în spațiul alocat unui vector nu încape adevărata rutină de serviciu; vectorii au adrese fixe; RST 0 coincide ca adresă de tratare cu adresa 0000 forțată prin inițializarea generală.

Din setul de circuite specializate asociate microprocesorului 8080 face parte circuitul 8259 de tratare a întreruperilor, PIC (Priority Interrupt Controller).

Descrierea sa va fi făcută în paragraful 6.9.3, mulțumindu-ne acum să enumerăm principalele lui avantaje și dezavantaje. Dintre avantaje citățm: există mai multe metode de acordare a priorității (fixă, prin rotație etc.) stabilite prin program; permite mascarea/demascarea separată a celor 8 niveluri de întrerupere admise; vectorii de întrerupere pot fi plasați oriunde în memorie utilizînd fiecare o zonă de 4 sau 8 octeți; permite extensia sistemului de întreruperi pînă la 64 de niveluri. Dezavantajele sale sînt: programarea complicată și dependența de circuitul specializat 8228 care-i furnizează cele trei impulsuri $\overline{\text{INTA}}$ necesare pentru forțarea pe magistrala de date a celor trei octeți ai instrucțiunii CALL care înlocuiește în acest caz instrucțiunea RST *n*.

6.9.2. PROGRAMAREA ÎNTRERUPERILOR ÎN CADRUL SD-8080

Logica de tratare a întreruperilor în cadrul SD-8080 a fost descrisă în paragraful 4.2.4. După cum ne reamintim sînt posibile trei cauze de întrerupere pe care le vom examina în ordinea priorității: recepționarea unui caracter pe interfața de modem, terminarea transmiterii unui caracter pe aceeași interfață și apăsarea cheii ÎNTRERUPERE de pe panoul de comandă. Schema folosită reprezintă o variantă ușor modificată a schemei de întreruperi cu prioritate software, reprezentată în figura 6.36*b*. În cazul întreruperilor modem sînt prezente toate componentele schemei cu excepția registrului de măști; mascarea uneia sau alteia dintre aceste întreruperi este posibilă prin dezactivarea emițătorului sau receptorului-serie din cadrul circuitului USART 8251. Întreruperea-panou intervine direct în disjuncția pentru realizarea semnalului de întrerupere INT; prezența ei e decelată implicit prin absența întreruperilor modem, nu are *latch* de intrare, fiind un semnal de tip nivel, nu poate fi mascată și nici nu are nevoie de achitare specifică întrucît luarea ei în considerare produce intrarea în monitor care dezactivează sistemul de întreruperi. Achitarea se face implicit, deoarece operatorul va înceta acționarea asupra cheii după ce constată intrarea în monitor. Unicul vector de întrerupere folosit este RST 7.

Organigrama servirii întreruperilor e prezentată în figura 4.27*c*. Programul aferent este dat mai jos.

```
; SECVENȚA DE TRATARE A ÎNTRERUPERILØR
        ØRG   38H     ;INTRAREA PENTRU RST 7
        PUSH  PSW     ;SALVARE CØNTEXT PRØCESØR
        PUSH  B
        PUSH  D
        PUSH  H
        IN    STINT   ;TEST DACA ÎNTRERUPERE MØDEM
        ANI   MDINT
        JNZ   MØDEM ;TRATAREA ÎNTRERUPERILØR MØDEM
        PØP   H       ;ESTE IT PANØU SAU BREAKPØINT
        PØP   D       ;REFACERE CØNTEXT
        PØP   B
        PØP   PSW
        JMP   MØNGØ ;INTRARE ÎN MØNITØR
MØDEM:  IN    STINT   ;TRATAREA IT MØDEM
        ANI   RXMØD
        JNZ   RCVR    ;ESTE IT RECEPȚIE?
        MVI   A,IACKT ;NU  ACHITARE IT EMISIE
        ØUT   MIACK
        LXI   H,RSTØR;ADRESA DE REFACERE
        PUSH  H       ;CØNTEXT ÎN STIVA
        LHLD  TSERV   ;SALT LA ADRESA DE
        PCHL          ;TRATARE DIN TSERV
RCVR:   MVI   A,IACKR ;ACHITARE IT RECEPȚIE
        ØUT   MIACK
        LXI   H,RSTØR;ADRESA DE REFACERE CØNTEXT
;ÎN STIVĂ PENTRU A CØNSTITUI PUNCTUL DE ÎNTØARCERE
;AL RUTINEI DE SERVICIU
        PUSH  H
        LHLD  RSERV ;SALT LA ADRESA DE TRATARE
;PØZIȚIØNATĂ DE PRØGRAMUL PRINCIPAL ÎN RSERV
        PCHL
RSTØR:  PØP   H       ;SECVENȚA DE REFACERE CØNTEXT
        PØP   D
        PØP   B
        PØP   PSW
        EI            ;REACTIVARE SISTEM DE ÎNTRERUPERI
        RET           ;REVENIRE ÎN PRØGRAMUL PRINCIPAL
;LA ADRESA UNDE EL A FØST ÎNTRERUPT.
MØNGØ   EQU   14H     ;ADRESA DE INTRARE EFECTIVĂ
                      ;MØNITØR
STINT   EQU   0F7H    ;ADRESA CUVÎNT DE STARE ÎNTRERU-
                      ;PERI
```

```
MDINT   EQU   4          ;MASCA ÎNTRERUPERE MØDEM
RXMØD   EQU   8          ;MASCA IT. RECEPȚIE MØDEM
TXMØD   EQU   10H        ;MASCA IT. EMISIE MØDEM
MIACK   EQU   0FDH       ;ADRESA ACHITARE ÎNTRERUPERI
IACKR   EQU   2          ;ACHITARE IT. RECEPȚIE MØDEM
IACKT   EQU   4          ;ACHITARE IT. EMISIE MØDEM
RSERV:  DW    SNØP       ;ADRESA DE INTRARE A RUTINEI
; DE SERVIRE SPECIFICĂ A IT. RECEPȚIE. MØDEM
TSERV   DW    SNØP       ;ADRESA DE INTRARE A RUTINEI
; DE SERVIRE SPECIFICĂ A IT. EMISIE MØDEM
; EXEMPLU DE RUTINĂ DE SERVIRE CARE FACE
; ÎNTRERUPEREA ÎN CHESTIUNE INEFECTIVĂ
SNØP:   RET              ;REVENIRE LA RSTØR FĂRĂ A FACE
                         ;NIMIC.
```

În secvența de program anterioară se pot distinge toate operațiile asociate rutinei de serviciu enumerate la paragraful anterior: salvare-context, achitare, tratare specifică, refacere context, reactivare sistem de întreruperi și revenire în programul principal. Este de asemenea exemplificată programarea ordinii de prioritate a întreruperilor în cadrul unui sistem nevectorizat prin ordinea de luare în considerare a biților din cuvîntul de stare a întreruperilor.

Punctul de intrare al fiecărei rutine de tratare specifică este furnizat de către programul principal rutinei generale de tratare a întreruperilor, prin intermediul a două zone de memorie RAM, RSERV și TSERV. În secvența prezentată, conținutul acestor zone este inițializat cu adresa unei rutine neefective, ceea ce dezactivează practic nivelurile respective. Atunci cînd programul principal, dorește schimbarea acțiunii asociate unei anumite cauze de întrerupere, el modifică conținutul locațiilor de legătură RSERV și TSERV. În cele ce urmează vom prezenta un exemplu de rutină de tratare specifică, ca și de program principal, folosind rutinele descrise.

```
; PRØGRAM PRINCIPAL
        .
        .                } Aici se face inițializarea USART modem, ambele rutine de tratare sînt dezactivate, pointerii RSERV și TSERV conținînd inițial adresa rutinei inefective SNØP
        .
        .
        LXI   H,RLINE ;INIȚIALIZARE RUTINA RECEPȚIE
                      ;MØDEM
        SHLD  RSERV
        .             } Prelucrări independente de conținutul liniei care se recepționează în bufferul de intrare
        .
TPNT:   LDA   LFULL   ;TESTARE INDICATØR LINIE CØMPLETĂ
        ØRA   A
        JZ    LABEL   ;E LINIE CØMPLETĂ?
        CALL  ANAL    ;DA, ANALIZĂ CØNȚINUT ȘI ACȚIUNE
```

```
        LXI   H,RLINE ;REACTIVARE RUTINA RECEPȚIE
                      ;MØDEM
        SHLD  RSERV
LABEL:  .             } Se continuă prelucrările deoarece nu s-a re-
        .             } cepționat o linie completă
; RUTINA DE TRATARE SPECIFICĂ IT. RECEPȚIE MØDEM
RLINE:  IN    MDATI   ;PRELUARE CARACTER
        ØUT   MDATØ   ;TRIMITERE ÎN ECØU
        CPI   CR      ;ESTE CR?
        JNZ   RLIN2
        MVI   A,LF    ;DA TRIMITE LF ÎN ECØU
        ØUT   MDATØ
RLIN1:  MVI   A,1     ;PØZIȚIØNARE INDICATØR DE RECEP-
                      ;ȚIØNARE
        STA   LFULL   ;LINIE CØMPLETĂ
        LXI   H,SNØP  ;DEZACTIVARE AUTØMATĂ A
        SHLD  RSERV   ;RUTINEI DE TRATARE IT. REC.
        RET
RLIN2:  LHLD  PNTR    ;ÎNCARCĂ PØINTER CURENT ÎN
        CPI   BS      ;BUFFERUL DE INTRARE
        JNZ   RLIN3
        DCX   H       ;ESTE BACK SPACE ELIMINĂ
        SHLD  PNTR    ;ULTIMUL CARACTER
        RET
RLIN3:  MØV   M,A     ;MEMØRARE CARACTER ÎN BUFFER
        INX   H       ;ACTUALIZARE PØINTER
        SHLD  PNTR
        RET
CR      EQU   0DH     ;CARRIAGE RETURN
LF      EQU   0AH     ;LINE FEED
BS      EQU   8       ;BACK SPACE
MDATI   EQU   5FH     ;INTRARE DATE USART MØDEM
MDATØ   EQU   5FH     ;IEȘIRE DATE USART MØDEM
LFULL:  DB    0       ;INDICATØR LINIE CØMPLETĂ INIȚIAL
                      ;=0
PNTR:   DW    BUFF    ;PØINTER ÎN BUFFER INTRARE
BUFF:   DS    80      ;BUFFER DE LINIE.
```

Se observă că după secvența de inițializare a adresei rutinei de tratare RLINE în pointerul RSERV programul principal continuă prelucrările, în timp ce rutina RLINE umple în mod asincron față de funcționarea programului principal bufferul de intrare BUFF. Rutina RLINE se autodezactivează la primirea unei linii complete, considerată a se termina cu CR. Sincronizarea între programul principal și rutina RLINE se face cu ajutorul indicatorului LFULL. Este exemplificat un punct de test al programului principal, TPNT, unde acesta inspectează indicatorul LFULL pentru a vedea

dacă s-a terminat recepționarea unei linii: în caz afirmativ, se dă controlul rutinei de analiză și execuție implicată de conținutul liniei, după care se reactivează rutina de tratare în vederea primirii unei noi linii. În caz negativ, prelucrările din unitatea centrală continuă și se va relua testarea indicatorului LFULL în alt punct, unde programul principal consideră că acest lucru este necesar. În punctul în care programul principal, prin natura algoritmului nu poate continua decît după recepționarea unei linii noi, se va introduce un test buclat al indicatorului LFULL.

Considerăm că acest exemplu poate ilustra în mod sugestiv economia de timp realizată într-un sistem de calcul lucrînd în întreruperi, unde este posibilă „suprapunerea" în timp a prelucrărilor din unitatea centrală cu transferurile de date la și de la dispozitivele periferice, cu toate că eliminarea totală a timpilor de așteptare ai unității centrale este imposibilă.

6.9.3. CIRCUITUL SPECIALIZAT DE TRATARE A ÎNTRERUPERILOR 8259

În schema din figura 6.37*a* este prezentat modul de conectare a circuitului 8259 la magistralele standard ale unui microsistem realizat cu ajutorul microprocesorului 8080. Din schemă rezultă în general și definițiile pinilor circuitului, așa că nu vom insista prea mult asupra acestui subiect, cititorul care dorește precizări suplimentare fiind rugat să consulte foaia de catalog a lui 8259. Liniile figurate în partea superioară a circuitului din schemă folosesc pentru conversația cu procesorul care constă din: programarea circuitului de către procesor și forțarea de către 8259 a vectorului de întrerupere în ciclurile $\overline{\text{INTA}}$. Reamintim (vezi și § 6.9.1) că pentru funcționarea lui 8259 este necesară prezența în schemă a circuitului 8228 (Data Bus Driver and System Controller), configurat astfel ca să furnizeze trei impulsuri $\overline{\text{INTA}}$ pentru fiecare ciclu ÎNTRERUPERE, permițînd astfel transmiterea vectorului de întrerupere ca o instrucțiune CALL la rutina de tratare corespunzătoare, ce poate fi plasată oriunde în memorie. Liniile CAS_0, CAS_1 și CAS_2 permit cascadarea circuitelor 8259, unui circuit PIC „master" putîndu-i fi atașate 8 alte asemenea circuite „slave", ceea ce permite extinderea sistemului la 64 de niveluri de întreruperi. Intrarea $\overline{\text{SP}}$ indică circuitului dacă va funcționa în mod „master" sau în mod „slave", iar IRQ_7—IRQ_0 sînt cererile de întrerupere, active pe frontul pozitiv. Linia IRQ_0 este asociată cererii de întrerupere celei mai prioritare.

Elementele esențiale ale schemei interne a circuitului 8259 sînt prezentate în figura 6.37*b*. Registrul IRR (Interrupt Request Register) corespunde *latch*-urilor de intrare din figura 6.36*a*; în el se memorează tranzițiile active, care apar pe oricare dintre liniile de cerere de întrerupere, IRQ_0—IRQ_7.

Dacă cel puțin unul dintre biții din IRR care este la un moment dat poziționat corespunde unui nivel de întrerupere remarcat prin intermediul bitului aferent din IMR (Interrupt Mask Register), atunci există condiții de generare a unei cereri de întrerupere pe linia INT. Lucrul acesta are loc însă numai dacă nivelul de prioritate al întreruperii în acel moment în serviciu,

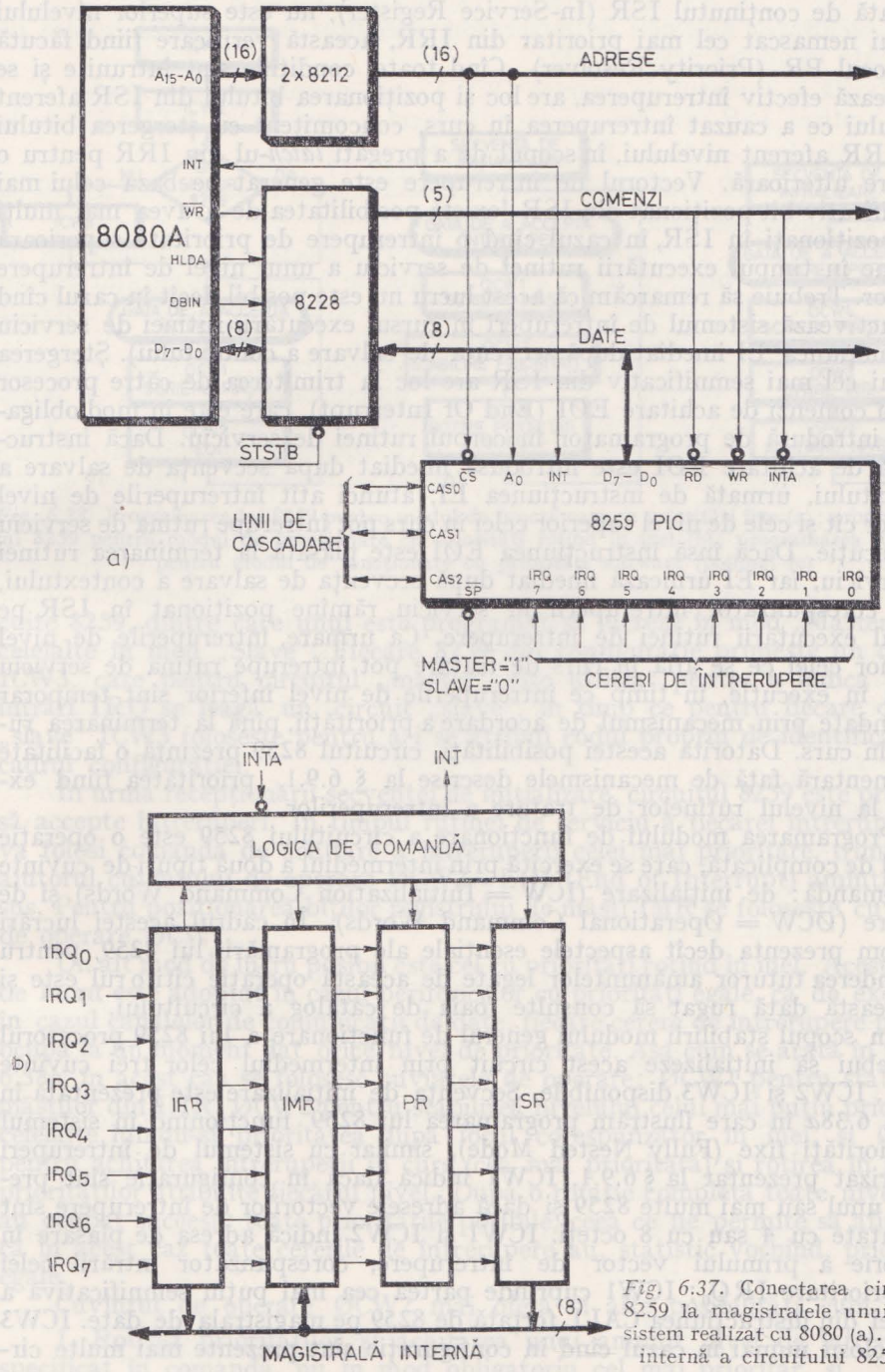


Fig. 6.37. Conectarea circuitului 8259 la magistralele unui microsistem realizat cu 8080 (a). Schema internă a circuitului 8259 (b)

indicată de conținutul ISR (In-Service Register), nu este superior nivelului bitului nemascat cel mai prioritar din IRR, această verificare fiind făcută de blocul PR (Priority Resolver). Cînd toate condițiile sînt întrunite și se generează efectiv întreruperea, are loc și poziționarea bitului din ISR aferent nivelului ce a cauzat întreruperea în curs, concomitent cu ștergerea bitului din IRR aferent nivelului, în scopul de a pregăti *latch*-ul din IRR pentru o folosire ulterioară. Vectorul de întrerupere este generat pe baza celui mai semnificativ bit poziționat din ISR; există posibilitatea de a avea mai mulți biți poziționați în ISR, în cazul cînd o întrerupere de prioritate superioară survine în timpul executării rutinei de serviciu a unui nivel de întrerupere inferior. Trebuie să remarcăm că acest lucru nu este posibil decît în cazul cînd se reactivează sistemul de întreruperi în cursul executării rutinei de serviciu (instrucțiunea EI imediat după secvența de salvare a contextului). Ștergerea bitului cel mai semnificativ din ISR are loc la trimiterea de către procesor a unei comenzi de achitare EØI (End Of Interrupt), care este în mod obligatoriu introdusă de programator în corpul rutinei de serviciu. Dacă instrucțiunea de achitare EØI este introdusă imediat după secvența de salvare a contextului, urmată de instrucțiunea EI, atunci atît întreruperile de nivel inferior cît și cele de nivel superior celei în curs pot întrerupe rutina de serviciu în execuție. Dacă însă instrucțiunea EØI este plasată la terminarea rutinei de serviciu, iar EI urmează imediat după secvența de salvare a contextului, bitul corespunzător întreruperii în serviciu rămîne poziționat în ISR pe timpul executării rutinei de întrerupere. Ca urmare, întreruperile de nivel superior celei ce se află în curs de tratare pot întrerupe rutina de serviciu aflată în execuție, în timp ce întreruperile de nivel inferior sînt temporar suspendate prin mecanismul de acordare a priorității, pînă la terminarea rutinei în curs. Datorită acestei posibilități, circuitul 8259 prezintă o facilitate suplimentară față de mecanismele descrise la § 6.9.1, prioritatea fiind extinsă la nivelul rutinelor de tratare a întreruperilor.

Programarea modului de funcționare a circuitului 8259 este o operație destul de complicată, care se exercită prin intermediul a două tipuri de cuvinte de comandă: de inițializare (ICW = Initialization Command Words) și de operare (ØCW = Øperational Command Words). În cadrul acestei lucrări nu vom prezenta decît aspectele esențiale ale programării lui 8259, pentru cuprinderea tuturor amănuntelor legate de această operație cititorul este și de această dată rugat să consulte foaia de catalog a circuitului.

În scopul stabilirii modului general de funcționare a lui 8259 procesorul va trebui să inițializeze acest circuit prin intermediul celor trei cuvinte ICW1, ICW2 și ICW3 disponibile. Secvența de inițializare este prezentată în figura 6.38*a* în care ilustrăm programarea lui 8259 funcționînd în sistemul cu priorități fixe (Fully Nested Mode), similar cu sistemul de întreruperi vectorizat prezentat la § 6.9.1. ICW1 indică dacă în configurație sînt prezente unul sau mai multe 8259 și dacă adresele vectorilor de întrerupere sînt distanțate cu 4 sau cu 8 octeți. ICW1 și ICW2 indică adresa de plasare în memorie a primului vector de întrerupere, corespunzător intrării celei mai prioritare IRQ_0; ICW1 cuprinde partea cea mai puțin semnificativă a adresei din instrucțiunea CALL forțată de 8259 pe magistrala de date. ICW3 este folosit numai în cazul cînd în configurație sînt prezente mai multe cir-

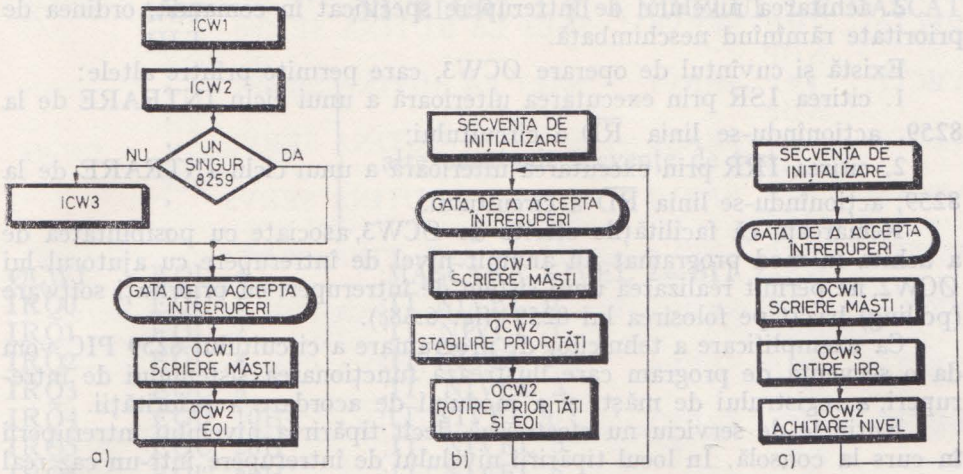


Fig. 6.38. Programarea lui 8259 pentru modul de funcționare cu priorități fixe (a); programarea lui 8259 pentru modul de funcționare cu priorități rotind în inel (b); programarea lui 8259 pentru modul de funcționare cu priorități software (polling) (c)

cuite 8259, dintre care unul este programat să lucreze în mod „Master", iar celelalte în mod „Slave". Fiecare 8259 din configurație primește un cuvînt ICW3, care pentru circuitul „master" este utilizat pentru a indica pe ce intrări IRQ se leagă un circuit „slave", în timp ce pentru fiecare circuit „slave" ICW3 folosește pentru a i se atribui codul propriu de identificare în cadrul configurației.

În urma recepționării secvenței de inițializare, circuitul 8259 este pregătit să accepte întreruperi. În timpul rutinei de serviciu a fiecărei întreruperi se va folosi comanda de achitare a întreruperii celei mai prioritare, trimisă cu ajutorul cuvîntului de operare ØCW2=EØI (End of Interrupt) amintit mai sus. Pentru poziționarea biților registrului de măști IMR se folosește cuvîntul de operare ØCW1.

Un alt mod de lucru posibil este cel cu priorități rotind în inel. Acest mod de lucru este adoptat în cazul perifericelor cu priorități egale, ca de exemplu în cazul canalelor de comunicații de date, fiecare cerere de întrerupere putînd ocupa la un moment dat orice nivel de prioritate. Așa cum se arată în figura 6.38*b*, în acest caz se folosesc cuvintele de operare: ØCW1 pentru stabilirea măștilor din IMR, ØCW2 pentru stabilirea nivelului celui mai puțin prioritar, celelalte luîndu-și prioritatea după locul corespunzător în inel, și ØCW2 pentru achitarea întreruperii în curs (cea mai prioritară) și rotirea în inel a priorităților atribuite fiecărui nivel. După o rotație completă toate nivelurile au ocupat succesiv toate prioritățile posibile, ceea ce ne permite să afirmăm că în acest caz toate cererile de întrerupere au, statistic vorbind, priorități egale.

Cuvîntul de operare ØCW2 mai oferă și alte posibilități cum ar fi:

1. Rotirea priorităților și achitarea unui anumit nivel de întrerupere specificat în comandă, nu în mod obligatoriu cel mai prioritar, și

2. achitarea nivelului de întrerupere specificat în comandă, ordinea de prioritate rămînînd neschimbată.

Există și cuvîntul de operare ØCW3, care permite printre altele:

1. citirea ISR prin executarea ulterioară a unui ciclu INTRARE de la 8259, acționîndu-se linia $\overline{RD}$ a circuitului;

2. citirea IRR prin executarea ulterioară a unui ciclu INTRARE de la 8259, acționîndu-se linia $\overline{RD}$ a circuitului.

Remarcăm că facilitățile oferite de ØCW3, asociate cu posibilitatea de a achita în mod programat un anumit nivel de întrerupere cu ajutorul lui ØCW2, ne permit realizarea unui sistem de întreruperi cu priorități software (polling) bazat pe folosirea lui 8259 (fig. 6.38*c*).

Ca exemplificare a tehnicilor de programare a circuitului 8259 PIC, vom da o secvență de program care ilustrează funcționarea sistemului de întreruperi, a registrului de măști și a modului de acordare a priorității.

Rutinele de serviciu nu efectuează decît tipărirea nivelului întreruperii în curs la consolă. În locul tipăririi nivelului de întrerupere, într-un caz real se va executa acțiunea specifică a rutinei respective: aducerea sau trimiterea unui caracter, actualizarea pointerilor și a indicatorilor de comunicație cu programul principal etc. Întreruperile sînt produse sub controlul programului principal care poate trimite impulsuri pe una sau mai multe dintre liniile IRQ, prin intermediul unei adrese de ieșire special afectate acestui scop. Programul principal este inițial stopat în HALT și execută succesiv diferite secvențe de test a întreruperilor, ca urmare a unei întreruperi de panou recepționate pe nivelul 7, cel mai puțin prioritar. Sistemul de întreruperi este programat în modul de lucru cu priorități fixe.

```
        ØRG 0
; PRØGRAMUL PRINCIPAL
; SECVENȚA DE INIȚIALIZARE SISTEM
        LXI  SP,4000H  ;INIȚIALIZARE PØINTER STIVĂ
        CALL INIT
; SECVENȚA DE TEST A ÎNTRERUPERILØR
        HLT            ;AȘTEAPTĂ IT PANØU PE NIVEL 7
TEST1:  MVI  A,IRQ0    ;IT. PE NIVEL 0
        ØUT  INTRP
        NØP            ;TIMP DE PRØPAGARE ÎNTRERUPERE
        HLT
TEST2:  MVI  A,IRQ0+IRQ1; IT SIMULTAN PE 2
        ØUT  INTRP     ;NIVELURI: 0 ȘI 1
        NØP
        HLT
TEST3:  MVI  A,ØCW1    ;MASCARE NIVEL 2
        ØUT  PICB
```

```
        MVI  A,IRQ2+IRQ4 ;IT. SIMULTAN PE 2
        NØP              ;NIVELURI: 2 ȘI 4, NIVELUL 2 E MASCAT
        HLT
          .
          .
          .              } alte eventuale secvențe de test
          .
          .
          .
ØCW1    EQU  4           ;MASCARE NIVEL 2=BIT 2
IRQ0    EQU  1           ;IT NIVEL 0
IRQ1    EQU  2           ;IT NIVEL 1
IRQ2    EQU  4           ;IT NIVEL 2
IRQ3    EQU  8           ;IT. NIVEL 3
IRQ4    EQU  10H         ;IT NIVEL 4
IRQ5    EQU  20H         ;IT. NIVEL 5
IRQ6    EQU  40H         ;IT. NIVEL 6
IRQ7    EQU  80H         ;IT. NIVEL 7
INTRP   EQU  0FEH        ;ADRESA TRIMITERE ÎNTRERUPERI,
                         ;CS=ABUS0
INIT:   MVI  A,ICW1      ;TRIMITERE ICW1
        ØUT  PICA        ;8259 CU ABUS0=0
        MVI  A,ICW2      ;TRIMITERE ICW2
        ØUT  PICB        ;8259 CU ABUS0=1
        EI               ;ACTIVARE SISTEM DE ÎNTRERUPERI
        RET              ;TØATE NIVELURILE SÎNT DEMASCATE
ICW1    EQU  16H
;ICW1:  BIT  0=0
;       BIT  1=1         UN SINGUR 8259
;       BIT  2=1         INTERVALUL DINTRE VECTØRI ESTE
;                        4 ØCT.
;       BIT  3=0
;       BIT  4=1
;BIT 7,6,5 = BIȚII CORESP. AI ADRESEI VECTØRULUI 0 (VECTØR
;=3000H)
ICW2    EQU  30H
;ICW2: PARTEA CEA MAI SEMNIFICATIVĂ A ADRESEI VECTØ-
;RULUI 0
;(VECTØR=3000H)
PICA    EQU  0FAH        ;ADRESA 8259 CU ABUS0=0, CS=ABUS2
PICB    EQU  0FBH        ;ADRESA 8259 CU ABUS0=1, CS=ABUS2
;ICW3: NEFØLØSIT, UN SINGUR 8259
        ØRG  3000H
```

```
; VECTØRII DE ÎNTRERUPERE PE 4 ØCTEȚI
VECT0:  JMP  SERV0    ;SALT LA RUTINA SERV0
        NØP
VECT1:  JMP  SERV1    ;SALT LA RUTINA SERV1
        NØP
VECT2:  JMP  SERV2    ;SALT LA RUTINA SERV2
        NØP
VECT3:  JMP  SERV3    ;SALT LA RUTINA SERV3
        NØP
VECT4:  JMP  SERV4    ;SALT LA RUTINA SERV4
        NØP
VECT5:  JMP  SERV5    ;SALT LA RUTINA SERV5
        NØP
VECT6:  JMP  SERV6    ;SALT LA RUTINA SERV6
        NØP
VECT7:  JMP  SERV7    ;SALT LA RUTINA SERV7
        NØP

; RUTINELE DE SERVICIU ALE ÎNTRERUPERILØR
SERV0:  CALL SAVE     ;SALVARE CØNTEXT
        CALL CRLF     ;SCRIERE CR,LF LA CØNSØLĂ
        MVI  C,30H    ;SCRIERE 0=NIVEL IT. LA CØNSØLĂ
        CALL CØ
        MVI  A,ØCW2   ;ACHITARE ÎNTRERUPERE=EØI
        ØUT  PICA
        CALL RSTR     ;REFACERE CØNTEXT
        EI            ;REACTIVARE SISTEM DE ÎNTRERUPERI
        RET
        .
        .
SERV7:  CALL SAVE     ;SALVARE CØNTEXT
        CALL CRLF     ;SCRIERE CR, LF LA CØNSØLĂ
        MVI  C,37H    ;SCRIERE 7=NIVEL IT. LA CØNSØLĂ
        CALL CØ
        MVI  A,ØCW2   ;ACHITARE ÎNTRERUPERE=EØI
        ØUT  PICA
        CALL RSTR     ;REFACERE CØNTEXT
        EI            ;REACTIVARE SISTEM DE ÎNTRERUPERI
        RET

; RUTINA DE SALVARE CØNTEXT
SAVE:   PUSH PSW
        PUSH B
        PUSH D
        PUSH H
        RET
```

```
; RUTINA DE REFACERE CØNTEXT
RSTR :  PØP  H
        PØP  D
        PØP  B
        PØP  PSW
        RET
ØCW2    EQU  20H        ;END—ØF—INTERRUPT
        END
```

După executarea secvenței de inițializare a circuitului 8259 și de activare a întreruperilor procesorul este stopat în starea HALT. În acest moment nici unul din cele opt niveluri de întrerupere nu este mascat. Pentru a intra în prima secvență de test este necesară producerea unei întreruperi-panou, care are rolul de a debloca procesorul. După executarea secvenței TEST1 la consolă s-a editat:

7 ca urmare a recepționării întreruperii panou;
0 ca urmare a producerii în mod programat a întreruperii de nivel 0, apoi procesorul rămîne din nou blocat în starea HALT.

La o nouă întrerupere-panou se tipărește:
7 cauzat de întreruperea panou;
0 întreruperea de nivel 0 mai prioritară;
1 întreruperea de nivel 1 mai puțin prioritară.

Deși ambele întreruperi au fost trimise simultan, ele au fost tratate de sistem în ordinea priorității lor. În secvența TEST3 se maschează nivelul 2, trimițîndu-se apoi întreruperea pe două niveluri: 2 și 4.

Se editează:

7 ca urmare a întreruperii panou ce amorsează secvența;
4 deoarece întreruperea de nivel 2, deși mai prioritară, este mascată, astfel că bitul corespunzător din IRR nu participă la calculul priorității și nici nu poate ajunge în ISR.

BIBLIOGRAFIE

1. DANCEA, I., *Microprocesoare*, Editura Dacia, Cluj-Napoca, 1979.
2. * * * *INTEL 8080 Microcomputer Systems User's Manual*
3. * * * *INTEL 8080 Assembly Language Programming Manual.*
4. PEATMAN, J., *Microcomputer-based Design*, Mc Graw-Hill, London, 1977.
5. BEASTON, J., *Using the 8259 Programmable Interrupt Controller*, AP-31, Intel Corp.
6. SMITH, L., *Using the 8251 USART*, AP-16, Intel Corp.
7. * * * *RMX/80 Real-Time Multitasking Executive*, AP-35, Intel Corp.
8. EBRIGHT, A., *8255 Programmable Peripheral Interface Applications*, Intel Corp., AP-15.
9. * * * *INTEL Data Catalog 1979, 1981.*
10. * * * *M6800 Application Manual*, Motorola Inc.
11. * * * *Peripheral Design Handbook*, Intel Corp.
12. GADWAY, J.R., *Principles of Data Acquisition and Conversion*, AN-79, Burr-Brown Research Corp., 1975.
13. DEJONG, M.L., e.a., *Microcomputer Interfacing: Characteristics of the 8253 Programmable Interval Timer*, Computer Design, feb. 1978.
14. * * * *2650 User's Manual*, vol. 1 și 2, Signetics Corp.

15. WEISSBERGER, A.J., *Data Communications Handbook*, Signetics Corp, 1977.
16. BLAKESLEE, T.R., *Digital Design with Standard MSI and LSI*, cap. 7 și 8, John Wiley and Sons, 1975.
17. SOUCEK, B., *Microprocessors and Microcomputers*, John Wiley and Sons Inc.. 1976.
18. FROHWERK, R.A., *Signature Analysis: A New Digital Field Service Method*, HP Journal, May 1977.
19. * * * *CCITT, Sixième Assemblée Plénière*, Genève, 27 Septembre — 6 Octobre 1976, Livre Orange, Tome VIII.2, Avis X.25.
20. ȚEPELEA, V., *Implementarea cuploarelor pentru calculatoarelor familiei FELIX folosind microprocesoare — cîteva considerații*, comunicare la sesiunea „Realizări și perspective în domeniul calculatoarelor electronice", ITC, București, 6—8 mai 1976.
21. ȚEPELEA, V.; KOVACS, A; PURICE, E., *SD-8080, un sistem de dezvoltare pentru microprocesoare*, comunicare la sesiunea „I.T.C. — 10 ani de activitate", București, 23—25 noiembrie 1978.
22. MOCANU, N; NĂSTASE, M; PĂUN, C.; ȚEPELEA, V.; KOVACS, A.; PURICE, E., *Dispozitiv programabil pentru analiza locală a măsurătorilor antropogamametrice și teleprelucrarea datelor spectrometrice cu ajutorul calculatorului FELIX C-256*, comunicare prezentată la a II-a ediție a sesiunii anuale de comunicări „Progrese în fizică", organizată de Institutul Central de Fizică, Cluj-Napoca, 19—21 iulie 1980.
23. ȚEPELEA. V.; KOVACS, A; PURICE, E.; DOGARU, A.; MOCANU, N.; NĂSTASE, M.; PĂUN, C., *Folosirea microcalculatoarelor pentru determinarea contaminării interne umane cu gamaemițători*, „Primul Simpozion Național de Teoria Sistemelor", Craiova, 14—15 noiembrie 1980, vol. I, p. 358—363.
24. MOISA, T., *Contribuții la proiectarea sistemelor de calcul cu prelucrare paralelă*, teză de doctorat, Facultatea Automatică, Institutul Politehnic București, 1981.
25. TOWNZEN, D., *A Task-Scheduling Executive Program for Microcomputer Systems*, Computer Design, June 1977, vol. 16. no. 6, p. 194—198.
26. * * * RMX/80™ USER'S GUIDE, Intel Corporation, 1979.
27. HØJBERG, S., *One-Step Programmable Arbiters for Multiprocessors*, Computer Design, April 1978, vol. 17, no. 4, p. 154—158.
28. SEIM, TH., *Numerical Interpolation for Microprocessor — Based Systems*, Computer Design, February 1978, vol. 17, no 2, p. 111—116.
29. LARSEN, D.; RONY, P.; DEJONG, M.; TITUS, C.; TITUS, J., *Microcomputer Interfacing: A Demonstration Program for the 8253 Timer*, Computer Design, March 1978, vol. 17, no. 3, p. 134—136.
30. * * * *ISIS-II Diskette Operating System*, Intel Corporation, 1977.

MICROPROGRAMARE

7.1. CONCEPTUL TRADIȚIONAL DE MICROPROGRAMARE

Conceptul de microprogramare, ca metodă de proiectare a structurilor de control numerice, este prezentat pentru prima dată în iulie 1951 de profesorul Maurice Wilkes de la Universitatea din Cambridge, la o conferință de calculatoare ținută la Universitatea din Manchester. În comunicarea sa [1], Wilkes enunța ideile principale care stau la baza microprogramării și propunea primul model (fig. 7.1) al unei structuri de control microprogramate:

„... să considerăm controlul propriu-zis, adică partea mașinii care generează impulsurile necesare validării porților asociate registrelor aritmetice sau de control. Proiectantul acestei părți din mașină acționează de obicei într-o manieră ad-hoc, proiectînd scheme-bloc pînă cînd ajunge la un aranjament care îi satisface cerințele tehnice și pare a fi acceptabil din punct de vedere economic. Aș dori să sugerez o cale prin care controlul poate fi proiectat mai *sistematic* (s.n.) și de aceea mai puțin complicat.

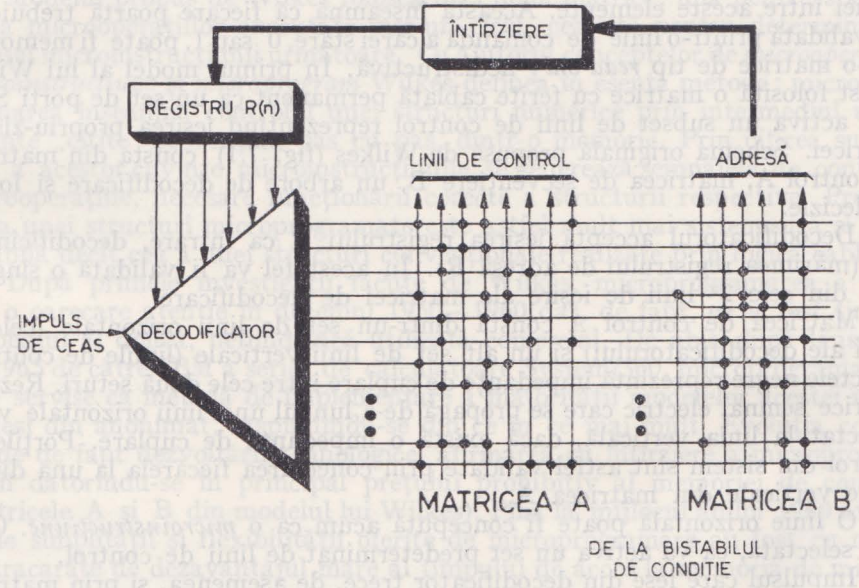


Fig. 7.1. Modelul structurii de control microprogramate propus de Maurice Wilkes

Orice operație apelată de un ordin care corespunde codului de ordin (instrucțiunii, n.n.) al mașinii presupune o *secvență de pași* (s.n.) ce poate cuprinde transferuri între memorie și registrele de *control* sau aritmetice și transferuri între registre. Fiecare din acești pași este realizat prin pulsarea unor anumite semnale asociate cu registrele de *control* sau aritmetice.

Voi numi acești pași *microoperații*. O operație-mașină propriu-zisă (instrucțiune, n.n.) este deci compusă dintr-o secvență sau un *microprogram* de microoperații.

Figura 4 (aici fig. 7.1,n.n.) reprezintă o schemă cu ajutorul căreia se obțin microoperațiile. Impulsul care generează o microoperație intră în arborele de decodificare și este condus la una din ieșiri în funcție de conținutul registrului R. El trece prin matricea A și generează impulsuri la unele din ieșirile matricei în funcție de aranjamentul impedanțelor de cuplaj. Aceste impulsuri validează porțile asociate registrelor de *control* sau aritmetice și deci realizează o microoperație. Impulsul generat în arborele de decodificare trece de asemenea și prin matricea B generînd impulsuri la unele din ieșirile acestei matrice. Aceste impulsuri sînt conduse printr-o scurtă linie de întîrziere la registrul R schimbînd conținutul acestuia. Ca rezultat, următorul impuls care intră în arbore va fi condus la alte ieșiri și în consecință va realiza o altă microoperație. Se vede deci că fiecare rînd din matricea A corespunde unei microoperații din secvența necesară (din microprogram, n.n.) realizării unei operații-mașină".

După cum se vede, Wilkes a observat că instrucțiunile-mașină pot fi realizate ca *secvențe de mai multe operații elementare* care să specifice transferul de informație între diferitele elemente ale unei unități centrale. Cu ajutorul unor porți de control proiectantul mașinii poate comanda circulația informației între aceste elemente. Aceasta înseamnă că fiecare poartă trebuie să fie validată printr-o linie de comandă a cărei stare, 0 sau 1, poate fi memorată într-o matrice de tip *read-only* nedistructivă. În primul model al lui Wilkes a fost folosită o matrice cu ferite cablată permanent ca un set de porți SAU care activa un subset de linii de control reprezentînd ieșirea propriu-zisă a matricei. Schema originală propusă de Wilkes (fig. 7.1) consta din matricea de control A, matricea de secvențiere B, un arbore de decodificare și logica de decizie.

Decodificatorul acceptă ieșirea registrului R ca intrare, decodificînd n biți (mărimea registrului de adresă R). În acest fel va fi validată o singură linie din cele 2^n linii de ieșire ale matricei de decodificare.

Matricea de control A constă dintr-un set de linii orizontale (cele 2^n ieșiri ale decodificatorului) și un alt set de linii verticale (liniile de control). Punctele negre reprezintă impedanțe de cuplare între cele două seturi. Rezultă că orice semnal electric care se propagă de-a lungul unei linii orizontale va fi conectat la linia verticală, dacă există o impedanță de cuplare. Porțile de control din sistem sînt astfel validate prin conectarea fiecăreia la una dintre liniile verticale din matricea A.

O linie orizontală poate fi concepută acum ca o *microinstrucțiune*. Cînd este selectată, ea va activa un set predeterminat de linii de control.

Impulsul care iese din decodificator trece, de asemenea, și prin matricea de secvențiere B. Liniile de ieșire ale acestei matrice poziționează un set de

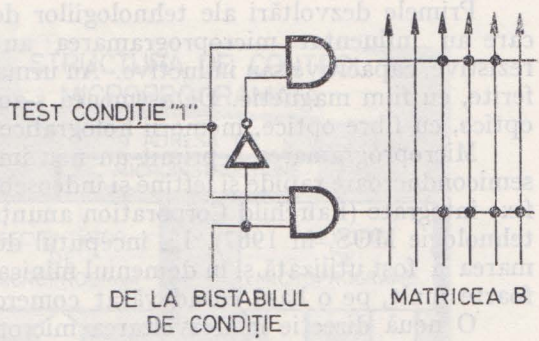


Fig. 7.2. Schema logică utilizată de Maurice Wilkes pentru execuția deciziei de secvențiere și selectarea adresei următoare

elemente de memorare care formează registrul de adresă R. Linia de întîrziere are rolul de a asigura prelungirea perioadei în care informația de control este stabilă, după schimbarea conținutului registrului de adresă R.

O microinstrucțiune va fi deci selectată corespunzător adresei din registrul de adresă R. Microinstrucțiunea va genera impulsuri de validare corespunzînd codului de control stocat în acel cuvînt și va selecta adresa microinstrucțiunii ce va controla starea mașinii în ciclul următor.

Logica de decizie și salt, necesară oricărei structuri de control, este asigurată de un bistabil de condiție, a cărui ieșire depinde de realizarea unui anumit test din sistem. În funcție de rezultatul testului (reușit sau nereușit) bistabilul de condiție selectează una din cele două căi alternative în microprogram. În figura 7.2 se dă schema logică utilizată pentru execuția deciziei de secvențiere și selectarea adresei microinstrucțiunii următoare.

Modelul propus de Wilkes folosea deci două matrice: una pentru specificarea microoperațiilor executate într-un ciclu, cealaltă pentru determinarea adresei microinstrucțiunii următoare. Cu ajutorul conceptelor de *microoperație*, *microinstrucțiune* și *microprogram* Wilkes definea în esență metoda: microprogramarea înseamnă controlul unei structuri numerice prin intermediul unor cuvinte „citite" secvențial, pas cu pas, dintr-o memorie. Prin citirea succesivă a acestor cuvinte, microinstrucțiunile, se generează semnalele de control, microoperațiile, necesare funcționării corecte a structurii respective. Proiectarea unei structuri microprogramate este astfel mult mai sistematică și mai flexibilă decît cea a unei structuri convenționale realizate prin logică cablată.

După primele investigații făcute de Wilkes, microprogramarea a primit o oarecare atenție în deceniul 1950—1960 dar, de fapt, ea a fost ținută pe planul al doilea, nefiind încă utilizată comercial. De abia după lansarea în 1965 de către IBM a seriei de calculatoare System/360, microprogramarea, care servise ca metodă de implementare a majorității modelelor acestei serii, va ieși din anonimat, răspîndindu-se din ce în ce mai mult. Evoluția corespunde de fapt dezvoltării tehnologice, afirmarea cu întîrziere a microprogramării datorîndu-se în principal prețului prohibitiv al memoriei de control (matricele A și B din modelul lui Wilkes). Pînă la mijlocul anilor 1960 avantajele simplității și flexibilității oferite de microprogramare au fost cu mult contracarate de dezavantajul mare al timpului de acces la memoria de microinstrucțiuni.

Primele dezvoltări ale tehnologiilor de fabricare a memoriei de control care au influențat microprogramarea au fost cele ale memoriilor liniare rezistive, capacitive sau inductive. Au urmat dispozitivele neliniare cu diode, ferite, cu film magnetic. De asemenea s-au realizat dispozitive de memorare optice, cu fibre optice, memorii holografice [4].

Microprogramarea a primit un nou impuls odată cu apariția memoriilor semiconductoare rapide și ieftine și îndeosebi după fabricarea primelor memorii fixe integrate (Fairchild Corporation anunța prima memorie fixă, realizată în tehnologie MOS, în 1967). La începutul deceniului 1970—1980 microprogramarea a fost utilizată și în domeniul minicalculatoarelor, răspîndindu-se astfel foarte mult, pe o bază cu adevărat comercială.

O nouă direcție în dezvoltarea microprogramării este marcată de realizarea circuitelor integrate pe scară largă. Este vorba de apariția memoriilor fixe LSI și mai ales a microprocesoarelor de tip *bit-slice* microprogramabile (în 1973 firma National Semiconductor lansează un circuit pe 4 biți, General Purpose Controller/Processor, care putea fi folosit ca element constructiv pentru lungimi de cuvînt de pînă la 32 de biți, iar firma Rockwell fabrică tot atunci un procesor paralel pe 4 biți, de asemenea microprogramabil, cu un set de 50 de instrucțiuni și un ciclu de 5 μs). Aceste progrese tehnologice au avut o influență deosebită asupra microprogramării, care a devenit din ce în ce mai mult o metodă modernă de proiectare a structurilor de control numerice, ieșind din sfera exclusivă a proiectanților de calculatoare și găsindu-și aplicații în cele mai diverse domenii. De aceea, în prezent, metoda a căpătat un interes general.

La sfîrșitul acestui paragraf introductiv trebuie să atragem atenția că datorită unei similarități de terminologie pot apărea confuzii în ceea ce privește *microprocesoarele* și *microprogramarea*. Termenul *microprocesor* este în general utilizat pentru a desemna un procesor LSI implementat într-un singur *chip*. Prin adăugarea altor circuite auxiliare un microprocesor poate fi extins la un *microcalculator*. Deci prefixul *micro* din termeni ca *microprocesor* și *microcalculator* se referă la dimensiunea fizică a dispozitivelor. Pe de altă parte, *microprogramarea* este o tehnică de proiectare a structurilor numerice și, de exemplu, în expresia procesor microprogramat/microprogramabil, ea se referă la o structură, la o anumită implementare, la o arhitectură, și nu trebuie confundată cu programarea unui microprocesor. Odată stabilită această distincție putem spune că există atît microprocesoare microprogramate, cît și microprocesoare microprogramabile.

7.2. CONCEPTUL MODERN DE MICROPROGRAMARE

7.2.1. O STRUCTURĂ DE CONTROL MICROPROGRAMATĂ

Deși microprogramarea a fost propusă ca o metodă de implementare a instrucțiunilor de limbaj-mașină, deci ca o metodă de proiectare a unității de control dintr-un calculator, conceptul de microprogramare a evoluat căpătînd un sens mai general. Pentru a ilustra această evoluție, vom da ca

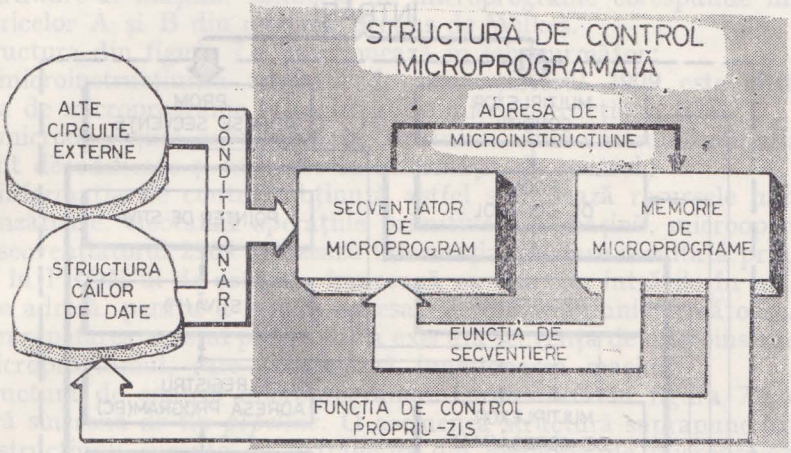


Fig. 7.3. O mașină microprogramată

exemplu o structură de control microprogramată realizată cu un secvențiator de adresă de tip 2909 (fig. 7.3 și 7.4).

În principiu, așa cum s-a văzut și în § 7.1, o *mașină microprogramată* este o mașină în care o secvență coerentă de microinstrucțiuni, deci un microprogram, este folosită pentru execuția operațiilor mari, macrooperațiilor, care definesc funcționarea mașinii. Dacă mașina este o unitate centrală a unui calculator, fiecare secvență de microinstrucțiuni va fi deci folosită pentru executarea unei (macro)instrucțiuni. Zona de memorare a acestor secvențe de microinstrucțiuni sau a microprogramelor, este de obicei numită *memorie de microprograme* sau *memorie de control*.

O mașină numerică se poate împărți în două părți care grupează căile de circulație a datelor, respectiv căile de control. Una din tehnicile de implementare a structurii căilor de control este și microprogramarea. În figura 7.3 se prezintă componența în mare a structurii microprogramate a căilor de control și relațiile globale cu restul mașinii numerice microprogramate. După cum se vede, structura de control microprogramată este alcătuită din două elemente principale: secvențiatorul de microprograme care generează adresa de microinstrucțiune și memoria de microprograme.

Structura de control microprogramată va avea două funcții principale: funcția de control propriu-zis și funcția de secvențiere. Conform acestora, microinstrucțiunea, elementul de control efectiv, are două părți care definesc de fapt *controlul prin microprogram*. Acestea se referă la:

— definirea și controlul tuturor *microoperațiilor* care trebuie realizate în mașină;

— definirea și controlul *adresei microinstrucțiunii următoare*.

Definirea diferitelor microoperații care trebuie executate în mașină cuprinde de obicei selecția operanzilor unității aritmetice-logice (UAL), funcția UAL, destinația UAL, controlul transportului, controlul deplasării, controlul

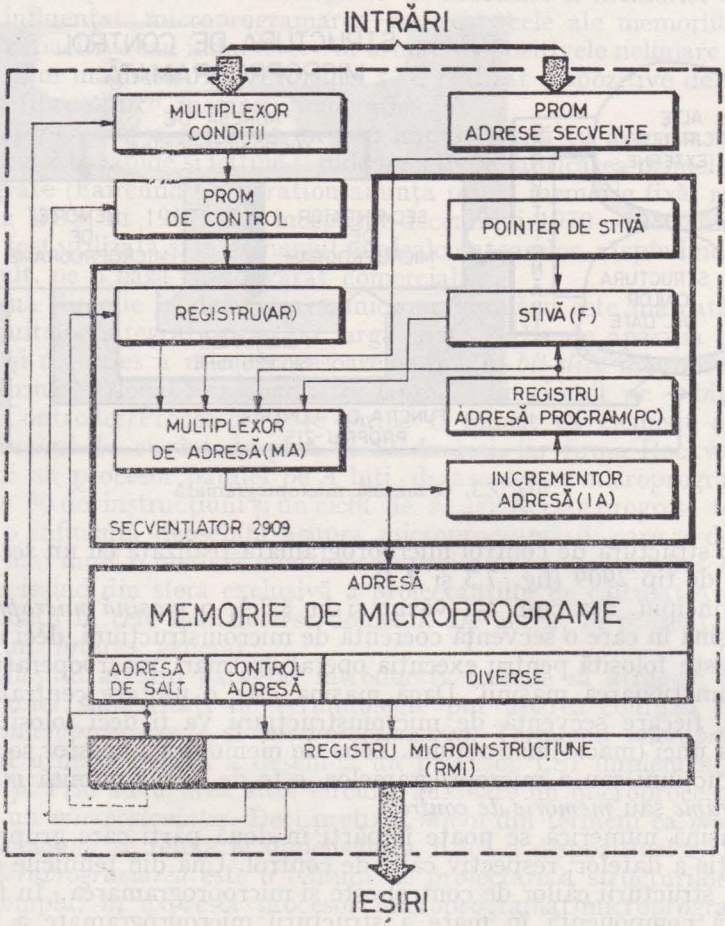


Fig. 7.4. O structură de control microprogramată realizată cu un secvențiator 2909

întreruperilor, controlul I/E, în general, controlul tuturor resurselor hardware ale mașinii.

Definirea adresei microinstrucțiunii următoare se referă la identificarea sursei pentru adresa următoare, la controlul condițiilor de test, uneori la generarea directă a valorii adresei.

Structura de control microprogramată poate fi, ca aceea din figura 7.4, alcătuită dintr-o memorie de microprograme, un registru de microinstrucțiune, un secvențiator de adresă de microprogram, o memorie de tip PROM pentru controlul acestui secvențiator, un multiplexor de condiții și o altă memorie PROM de *mapare* pentru generarea adreselor de început ale secvențelor corespunzătoare macrooperațiilor pe care trebuie să le execute mașina. Memoria de microprograme conține secvențe de microinstrucțiuni, compuse la rîndul lor din microoperații reprezentînd operațiile primitive ce trebuie să le reali-

zeze hardware-ul mașinii. Memoria de microprograme corespunde în acest fel matricelor A și B din modelul propus de Wilkes.

Structura din figura 7.4 funcționează în felul următor:

— microinstrucțiunea adresată de secvențiatorul 2909 este citită din memoria de microprograme în registrul de microinstrucțiune RMI;

— microoperațiile care intră în alcătuirea microinstrucțiunii aflate în RMI sînt decodificate și utilizate ca informație de control;

— informația de control obținută astfel stimulează resursele hardware corespunzătoare, efectuînd operațiile primitive din mașină, microoperațiile;

— secvențiatorul 2909 utilizează informația de stare rezultată prin decodificare în PROM-ul de control, împreună cu una din intrările în multiplexorul de adresă, pentru a genera adresa microinstrucțiunii următoare.

Prin repetarea acestui proces se va executa secvența de microinstrucțiuni, deci microprogramul, care controlează funcționarea mașinii.

Structura de control microprogramată prezentată în figura 7.4 este o structură sincronă de tip *pipeline*. O asemenea structură suprapune execuția microinstrucțiunii curente cu extragerea din memoria de microprograme a microinstrucțiunii următoare. Elementul-cheie al unei astfel de structuri este registrul *pipeline* (notat RMI în figura 7.4) plasat la ieșirea memoriei de microprograme. Această tehnică, utilizată pentru îmbunătățirea performanțelor de timp ale unei structuri de control microprogramate, va fi detaliată în § 7.2.2.

Secvențiatorul 2909, așa cum se vede în figura 7.4, este alcătuit dintr-un multiplexor de adresă MA cu 4 intrări, un registru AR, stiva pentru bucle și subrutine F, registrul-adresă-program PC și incrementorul de adresă IA. Se observă că adresa următoare poate fi selectată cu ajutorul multiplexorului MA care are următoarele intrări: intrarea directă D, registrul AR, registrul program PC și stiva F. Structura și funcționarea secvențiatorului 2909 vor fi descrise amănunțit în Cap. 9. Totuși, pentru înțelegerea complexității funcției de secvențiere a unei structuri microprogramate vom menționa că, dacă pentru controlul adresei următoare se folosește un cîmp de 3 biți, cu ajutorul acestuia pot fi implementate următoarele microoperații:

CONTINUARE	adresa următoare este adresa precedentă incrementată cu 1;
SALT	adresa următoare este adresa din registrul AR;
SALT PE CONDIȚIE	adresa următoare este adresa din registrul AR, atunci cînd condiția selectată este adevărată; în caz contrar, secvența continuă;
SALVARE PC	adresa din PC este introdusă în stivă;
SALT LA SUBRUTINĂ	adresa următoare este adresa din registrul AR, adresa curentă este memorată în stivă;
ADRESĂ DE START	adresa următoare este adresa generată de PROM-ul pentru adrese de secvențe (fig. 7.4);
REVENIRE	adresa următoare este adresa memorată ultima oară în stivă;
TEST SFÎRȘIT BUCLĂ	adresa următoare este adresa memorată ultima oară în stivă cînd condiția selectată este adevărată; în caz contrar, secvența continuă.

Folosind aceste microoperații de secvențiere se pot scrie ușor microprograme destul de complicate necesare controlului unor diverse mașini numerice.

Structura microprogramată descrisă mai sus poate fi privită și ca o *cutie neagră* ale cărei intrări sînt macrooperațiile și condițiile de test, iar ieșirile — cîmpurile de control din microinstrucțiune. Funcțiile pe care le poate realiza, cea de control propriu-zis și cea de secvențiere, sînt definite prin microprogramul stocat în memoria de microprograme. O astfel de structură, pe care o vom numi mai departe *structură de control microprogramată*, SCM, poate fi utilizată pentru implementarea funcției de control din orice mașină numerică, bineînțeles cu particularitățile, avantajele și dezavantajele ce vor fi discutate în capitolele următoare.

Evoluînd astfel spre generalitate, *o structură de control microprogramată* poate fi definită ca un automat finit destinat implementării funcției de control din sisteme care prelucrează informația, folosind tehnica microprogramării. O asemenea structură, al cărei prim model a fost propus de Maurice Wilkes, se caracterizează în general prin:

— organizarea memoriei de control;

— formatul microinstrucțiunii;

— implementarea microinstrucțiunii.

În continuare se vor prezenta pe larg aceste caracteristici ale SCM.

7.2.2. CARACTERISTICI ALE STRUCTURILOR DE CONTROL MICROPROGRAMATE

7.2.2.1. **Organizarea memoriei de microprograme**

S-a definit mai sus că memoria de microprograme sau memoria de control este o unitate de memorie de mare viteză în care se găsesc microprogramele în execuție. Această memorie este de două feluri: de tip ROS, *Read Only Storage*, sau ROM, *Read Only Memory*, și de tip WCS, *Writable Control Store*. Memoria ROS este fixă pentru o structură dată și nu poate fi modificată sub controlul microprogramului. Schimbarea conținutului memoriei ROS este deci echivalentă, sub aspect funcțional, cu schimbarea structurii. În cazul memoriei WCS, conținutul ei poate fi modificat sub controlul microprogramului. Astfel, cu ajutorul unei memorii WCS, o SCM poate fi modificată funcțional pe o bază dinamică.

Memoria de microprograme a unei SCM, indiferent de caracterul ei ROS și/sau WCS, poate fi organizată logic în mai multe moduri (figura 7.5).

1. Cea mai simplă și mai obișnuită structură de memorie de microprograme este cea în care fiecărui cuvînt de memorie îi corespunde o singură microinstrucțiune (fig. 7.5*a*). Citirea unei microinstrucțiuni presupune un singur acces la memoria de control.

2. O altă structură de memorie de microprograme este aceea în care fiecare cuvînt de memorie conține două sau mai multe microinstrucțiuni (figura 7.5*b*). Avantajul acestei scheme este că în registrul de microinstrucțiune

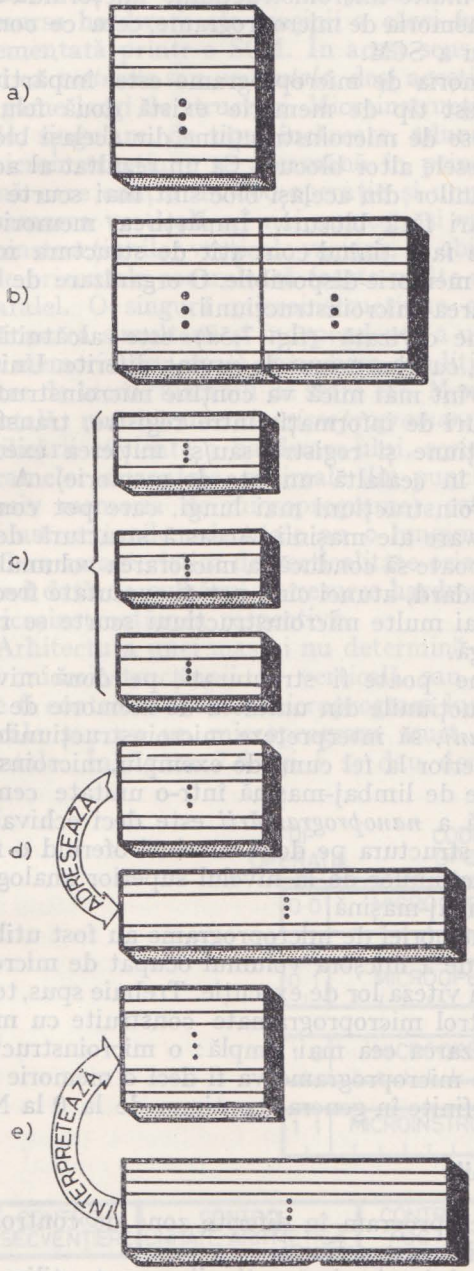


Fig. 7.5. Organizări ale memoriei de microprograme

se citesc simultan două sau mai multe microinstrucțiuni, micșorîndu-se în acest fel numărul de referințe la memoria de microprograme, ceea ce conduce în final la mărirea vitezei de lucru a SCM.

3. Într-o altă structură, memoria de microprograme este împărțită în blocuri (figura 7.5*c*). Pentru acest tip de memorie există două feluri de adrese de microinstrucțiune: adrese de microinstrucțiune din același bloc cu microinstrucțiunea curentă și adresele altor blocuri. Ca un rezultat al acestei organizări adresele microinstrucțiunilor din același bloc sînt mai scurte decît cele corespunzătoare unei structuri fără blocuri. Împărțirea memoriei de microprograme pe blocuri se poate face ținînd cont atît de structura microprogramului, cît și de circuitele de memorie disponibile. O organizare de acest fel conduce în general la micșorarea microinstrucțiunii.

4. Memoria de microprograme divizată (fig. 7.5*d*) este alcătuită din două unități de memorie distincte, cu dimensiuni de cuvînt diferite. Unitatea de memorie cu dimensiune de cuvînt mai mică va conține microinstrucțiuni mai simple (de exemplu, transferuri de informație între registre, transferuri de informație între microinstrucțiune și registre sau/și inițierea execuției unei microinstrucțiuni rezidente în cealaltă unitate de memorie). A doua unitate de memorie conține microinstrucțiuni mai lungi, care pot controla simultan mai multe resurse hardware ale mașinii. Această structură de memorie de microprograme divizată poate să conducă la micșorarea volumului de memorie față de o organizare standard, atunci cînd pot fi executate frecvent microinstrucțiuni scurte sau/și mai multe microinstrucțiuni scurte se referă la aceeași microinstrucțiune lungă.

5. Memoria de microprograme poate fi structurată pe două niveluri (fig. 7.5*e*), astfel încît microinstrucțiunile din unitatea de memorie de nivel mai scăzut, numite *nanoinstrucțiuni*, să interpreteze microinstrucțiunile din unitatea de memorie de nivel superior la fel cum, de exemplu, microinstrucțiunile interpretează instrucțiunile de limbaj-mașină într-o unitate centrală microprogramată. Această tehnică a *nanoprogramării* este deci echivalentă conceptual cu microprogramarea, structura pe două niveluri oferind o flexibilitate în proiectarea microinstrucțiunilor de la nivelul superior analogă cu proiectarea instrucțiunilor de limbaj-mașină.

Toate aceste organizări ale memoriei de microprograme au fost utilizate în minicalculatoare [2] cu scopul de a micșora volumul ocupat de microprogramele de control sau/și de a mări viteza lor de execuție. Trebuie spus, totuși, că, în general, structurile de control microprogramate construite cu microprocesoare *bit-slice* folosesc organizarea cea mai simplă: o microinstrucțiune pe cuvînt (fig. 7.5*a*). Memoria de microprograme va fi deci o memorie de N cuvinte a cîte M biți cu locații definite în general continuu de la 0 la N—1.

7.2.2.2. **Formatul microinstrucțiunii**

Împărțirea cuvîntului de microprogram în diferite zone de control numite *cîmpuri* definește *formatul* microinstrucțiunii.

Pentru caracterizarea formatului microinstrucțiunii se pot utiliza mai mulți parametri: caracteristica de verticalitate-orizontalitate, modul de control, gradul de codificare.

Deosebit de important în proiectarea microinstrucțiunii este numărul de resurse hardware ale mașinii a cărei funcție de control se dorește să fie implementată printr-o SCM. În acest sens microinstrucțiunile se clasifică de obicei ca *verticale* sau *orizontale*, deși acești termeni se referă la extremitățile unei game largi de structuri. Microinstrucțiunile verticale efectuează operații simple, singulare, de tipul încărcare, adunare, memorare, salt etc. Acest tip de microinstrucțiune se aseamănă în principiu cu instrucțiunile de limbaj-mașină care conțin un cod-operație și unul sau mai mulți operanzi (microprogramarea verticală se mai numește și *microprogramare soft* [3]). Lungimea microinstrucțiunilor verticale variază, de obicei, între 12—24 biți. Microinstrucțiunile orizontale controlează foarte multe resurse hardware care funcționează în paralel. O singură microinstrucțiune orizontală ar putea să controleze funcționarea simultană și independentă a uneia sau a mai multor UAL, accesul la memoria principală, generarea condiționată a adresei următoare, diverse registre de lucru, bistabili de stare etc. Menționăm că, deși microprogramarea orizontală, numită uneori *microprogramare hard* [3], are avantajul potențial al utilizării eficiente a hardware-ului, scrierea și punerea la punct a microprogramelor orizontale, optimale din punctul de vedere al utilizării resurselor (inclusiv memoria de microprograme), este o problemă destul de dificilă. Microinstrucțiunile orizontale au o lungime tipică de 64 biți. Așadar, vom considera caracteristica de verticalitate-orizontalitate a unei microinstrucțiuni ca fiind dată de numărul de resurse hardware din mașină controlate *simultan* de microinstrucțiunea respectivă.

Arhitectura unei mașini nu determină în mod necesar și complet proiectarea microinstrucțiunii ca verticală sau orizontală: o aceeași arhitectură poate fi controlată cu un microprogram lung, lent, scris cu microinstrucțiuni verticale sau cu un microprogram scurt, rapid, scris cu microinstrucțiuni orizontale. În figura 7.6*a* și *b* se dau două formate de microinstrucțiune,

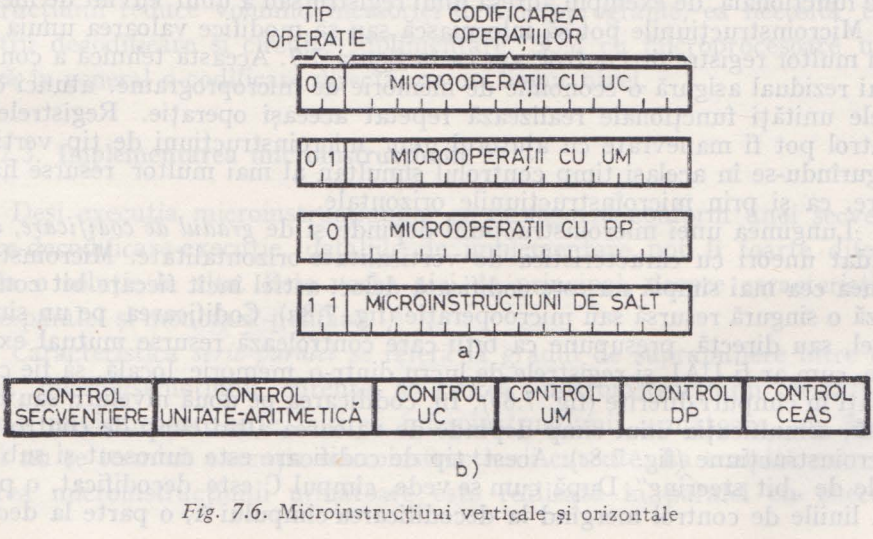


Fig. 7.6. Microinstrucțiuni verticale și orizontale

vertical respectiv orizontal, pentru controlul unui canal selector microprogramat (cap. 10, fig. 10.4).Repertoriul de microinstrucțiuni verticale cuprinde patru tipuri de operații codificate într-un cîmp de doi biți. Aceste tipuri de operații mari corespund controlului prin microprogram al uneia din principalele resurse hardware ale mașinii, în acest caz blocurile de logică cu UC, UM, DP și structura de control SCM. Microoperațiile, specifice tipului de operație, pot fi codificate corespunzător în restul de biți ai microinstrucțiunii. Microinstrucțiunea orizontală propusă este împărțită în cîmpuri separate pentru controlul *simultan* al principalelor resurse hardware ale mașinii.

Dezvoltarea continuă în domeniul tehnologiei circuitelor oferă proiectantului de structuri de control blocuri constructive din ce în ce mai complexe. Această situație face ca puterea unei microoperații să fie într-o permanentă creștere, astfel încît ceea ce mai demult putea fi considerat ca format din mai multe microoperații distincte să poată fi înglobat azi într-o singură microoperație. Din acest motiv linia de despărțire între microinstrucțiuni verticale și orizontale nu mai poate fi bine definită. Nu lipsit complet de umor, termenul microinstrucțiuni *diagonale* salvează această situație. Astfel de microinstrucțiuni combină cele mai bune caracteristici ale microinstrucțiunilor verticale și orizontale: sînt ușor de înțeles, generat și implementat, ca și microinstrucțiunile verticale, avînd în același timp o capacitate crescută de control simultan al resurselor hardware ale mașinii.

Microinstrucțiunile descrise pînă acum au ilustrat *controlul imediat:* microoperațiile sînt convertite în semnale de control care acționează direct și imediat resursele mașinii. O altă tehnică de microprogramare, numită *control rezidual,* folosește unele registre preîncărcate pentru controlul resurselor hardware. Într-o schemă cu control rezidual microoperațiile nu mai controlează direct resursele. Acest rol va fi îndeplinit de registrele de control încărcate anterior. Valoarea unui asemenea registru preîncărcat, controlul rezidual, poate reprezenta microoperația pe care trebuie să o realizeze o anumită unitate funcțională, de exemplu adresa unui registru sau a unui cuvînt de memorie. Microinstrucțiunile pot să înlocuiască sau să modifice valoarea unuia sau mai multor registre de control, așa ca în figura 7.7. Această tehnică a controlului rezidual asigură o economie de memorie de microprograme, atunci cînd unele unități funcționale realizează repetat aceeași operație. Registrele de control pot fi manevrate cu ajutorul unor microinstrucțiuni de tip vertical, asigurîndu-se în același timp controlul simultan al mai multor resurse hardware, ca și prin microinstrucțiunile orizontale.

Lungimea unei microinstrucțiuni depinde și de *gradul de codificare,* confundat uneori cu caracteristica de verticalitate-orizontalitate. Microinstrucțiunea cea mai simplă nu este codificată deloc, astfel încît fiecare bit controlează o singură resursă sau microoperație (fig. 7.8*a*). Codificarea pe un singur nivel, sau directă, presupune ca biții care controlează resurse mutual exclusive, cum ar fi UAL și registrele de lucru dintr-o memorie locală, să fie combinați în cîmpuri diferite (fig. 7.8*b*). În codificarea pe două niveluri, sau indirectă, semnificația unui cîmp depinde de valoarea altui cîmp de control din microinstrucțiune (fig. 7.8*c*). Acest tip de codificare este cunoscut și sub numele de „bit steering". După cum se vede, cîmpul C este decodificat, o parte din liniile de control mergînd la decodificarea cîmpului A, o parte la decodi-

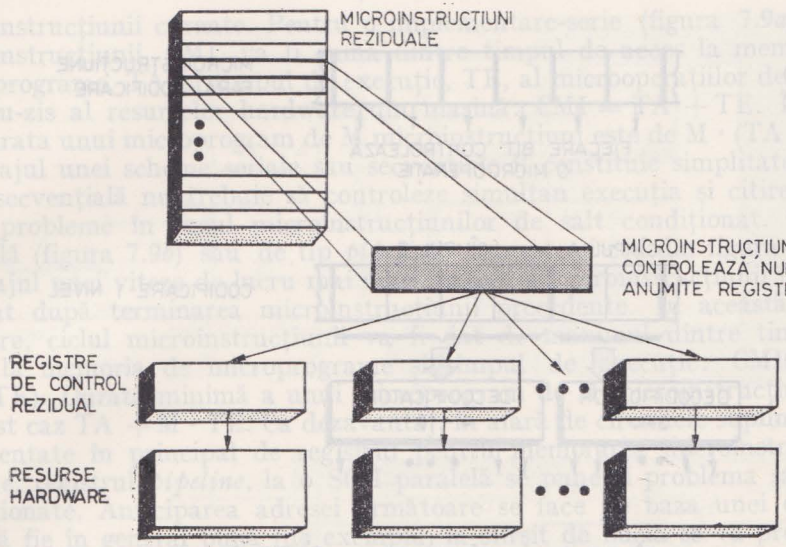


Fig. 7.7. Control rezidual

ficarea cîmpului B. În acest fel se poate mări puterea de control a unor cîmpuri din microinstrucțiune. O altă variantă a acestei tehnici este prezentată în figura 7.8*d*. Aici același cîmp controlează două resurse hardware cu funcționare separată în timp. „Dirijarea" (steering) cîmpului se face cu ajutorul unui bit de control. Într-un alt tip de codificare pe două niveluri, numită comutare de format, „format shifting", formatul microinstrucțiunii depinde de starea mașinii (de exemplu, pentru o unitate centrală, efectuarea unei operații de I/E sau prelucrarea unei instrucțiuni). Deși codificarea microinstrucțiunii reduce volumul memoriei de microprograme, ea necesită timp pentru decodificare și circuite suplimentare. SCM cu microprocesoare utilizează în general o codificare directă, pe un singur nivel.

7.2.2.3. Implementarea microinstrucțiunii

Deși execuția microinstrucțiunilor se desfășoară conform unei secvențe citire-decodificare-execuție, detaliile de implementare pot fi foarte diferite de la o soluție la alta. Este vorba aici, în principal, despre caracteristicile serie-paralel și monofază-polifază.

Caracteristica *serie-paralel* se referă la gradul de suprapunere între execuția microinstrucțiunii curente și citirea microinstrucțiunii următoare. Într-o implementare-serie, citirea microinstrucțiunii următoare nu începe pînă nu se termină execuția microinstrucțiunii curente. La cealaltă extremă, citirea microinstrucțiunii următoare este realizată în paralel cu execuția

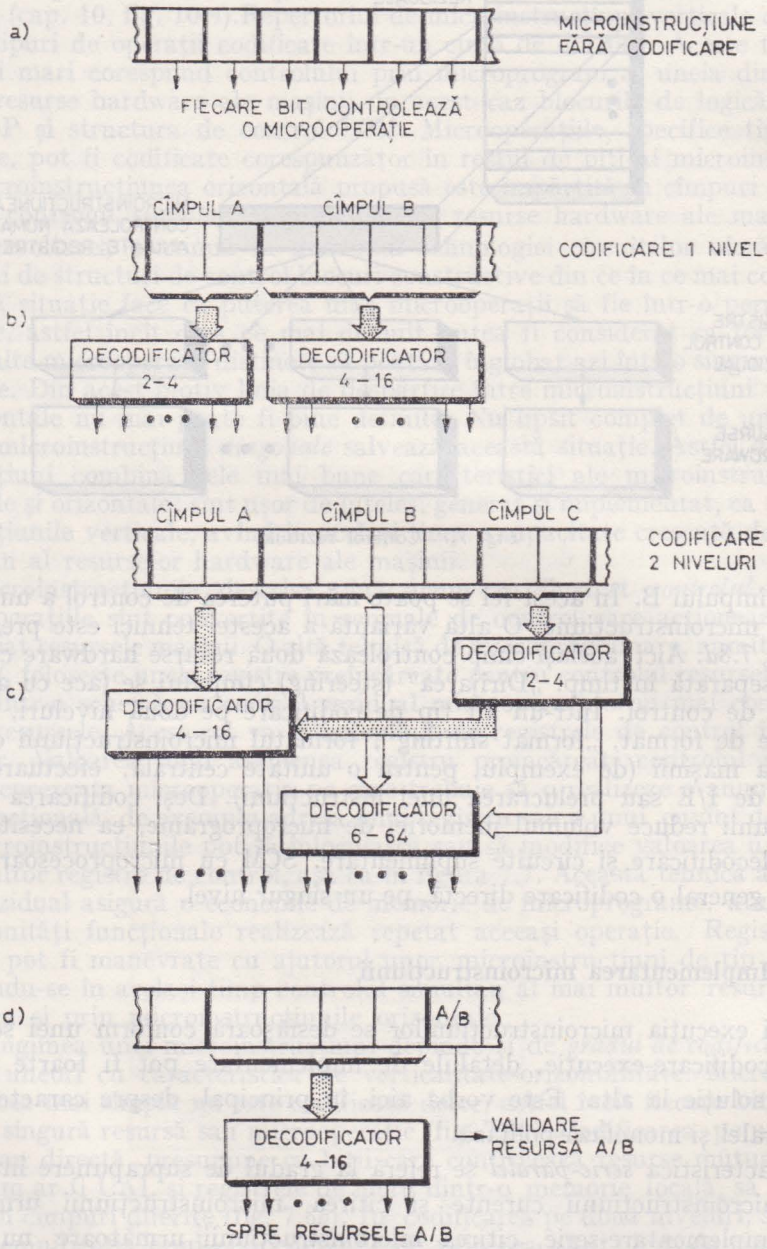


Fig. 7.8. Moduri de codificare a microinstrucțiunii

microinstrucțiunii curente. Pentru o implementare-serie (figura 7.9*a*), ciclul microinstrucțiunii, CMI, va fi suma dintre timpul de acces la memoria de microprograme, TA, și timpul de execuție, TE, al microoperațiilor de control propriu-zis al resurselor hardware din mașină: CMI = TA + TE. În acest fel, durata unui microprogram de M microinstrucțiuni este de M · (TA + TE). Avantajul unei scheme seriale sau secvențiale îl constituie simplitatea ei. O SCM secvențială nu trebuie să controleze simultan execuția și citirea și nu pune probleme în cazul microinstrucțiunilor de salt condiționat. O SCM paralelă (figura 7.9*b*) sau de tip *pipeline*, cum este și cea din figura 7.4, are avantajul unei viteze de lucru mai mari, execuția microinstrucțiunii începînd imediat după terminarea microinstrucțiunii precedente. În această implementare, ciclul microinstrucțiunii va fi dat de maximul dintre timpul de acces la memoria de microprograme și timpul de execuție: CMI = max (TA, TE). Durata minimă a unui microprogram de M microinstrucțiuni este în acest caz TA + M · TE. Ca dezavantaj, în afară de circuitele suplimentare, reprezentate în principal de registrul pentru memorarea microinstrucțiunii curente, registrul *pipeline*, la o SCM paralelă se pune și problema salturilor condiționate. Anticiparea adresei următoare se face pe baza unei estimări care să fie în general bună (de exemplu, la sfîrșit de buclă se va presupune că bucla se reia—estimarea cea mai probabilă). Dacă anticiparea este corectă, majoritatea cazurilor, nu se va pierde din viteză. În cazul unei anticipări greșite se va pierde însă cel puțin un ciclu pentru citirea corectă a microinstrucțiunii ce urmează.

Caracteristica *monofază-polifază* se referă la numărul de faze (cicli secundari, subcicli) utilizate într-un ciclu principal, pentru execuția unei microinstrucțiuni. Într-o implementare monofazată nu există cicli secundari ai ciclului de microinstrucțiune principal și microinstrucțiunea este efectivă o singură perioadă de ceas, toate semnalele de control fiind generate simultan (fig. 7.10*a*). Într-o implementare polifazată fiecare ciclu de ceas principal cuprinde mai mulți cicli secundari. Semnalele de control sînt generate secvențial corespunzător fiecărei faze (fig. 7.10*b*). Controlul ciclului microinstrucțiunii se poate face cu ajutorul unui cîmp care să fazeze în timp semnalele de control utilizate. În figura 7.10*c* este dată o altă variantă în care ciclul microinstrucțiunii este modificat de la o microinstrucțiune la alta în funcție de timpul de execuție. Acest ciclu variabil poate fi și polifazat (figura 7.10*d*).

7.3. AVANTAJELE ȘI DEZAVANTAJELE MICROPROGRAMĂRII

După trecerea în revistă a principalelor caracteristici ale microprogramării, ca metodă de proiectare a structurilor de control numerice, vom prezenta unele din avantajele și dezavantajele acestei metode.

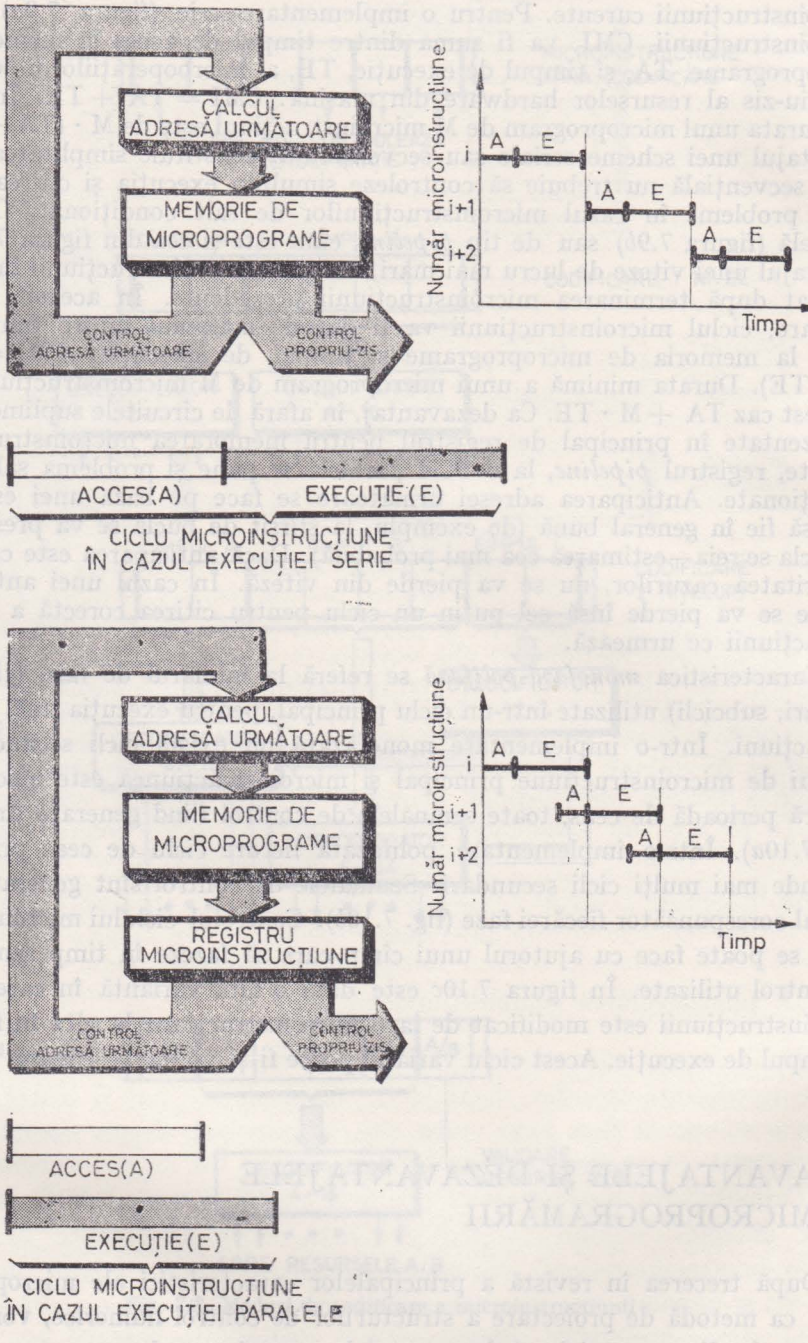


Fig. 7.9. Execuția serială și paralelă a microinstrucțiunii

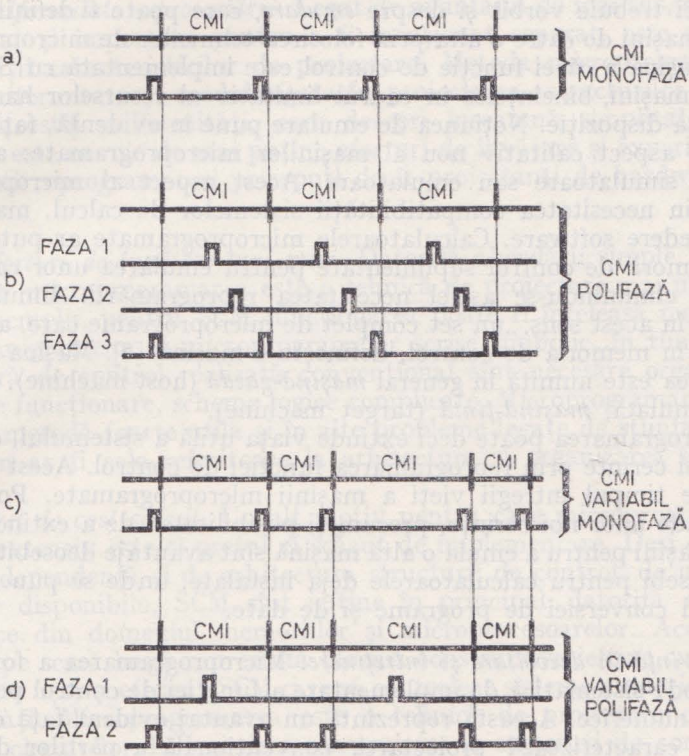


Fig. 7.10. Tipuri ale ciclului de microinstrucțiune (CMI)

7.3.1. AVANTAJELE MICROPROGRAMĂRII

1. *Flexibilitate, adaptabilitate.* Se știe că una din caracteristicile pe care trebuie să le aibă structurile de control pentru a fi cît mai eficiente este flexibilitatea modului de control (v. cap. 2). Această cerință poate fi asigurată de structurile de control programabile, din rîndul cărora fac parte și SCM. Divizarea proiectului unei SCM într-un proiect de hardware și un proiect de firmware permite ca microcodul să fie finalizat tîrziu în ciclul de proiectare, microprogramarea și realizarea hardware a mașinii efectuîndu-se în paralel. Acest paralelism în proiectare oferă arhitecților de sistem posibilitatea să efectueze relativ comod modificări ulterioare de arhitectură. Prin modificarea microprogramelor (și microprogramare dinamică) SCM se pot adapta ușor la noi interfețe de proces.

Flexibilitatea SCM este dată și de posibilitatea lor de a-și reorganiza resursele hardware și căile de circulație a informației pentru a „traduce", interpreta, intrările în ieșiri și funcțiuni, prin intermediul microprogramelor de control.

Tot aici trebuie vorbit și despre *emulare*, care poate fi definită ca simularea unei mașini de către o alta prin folosirea tehnicilor de microprogramare. Astfel, o mașină a cărei funcție de control este implementată cu SCM poate emula alte mașini, bineînțeles în cadrul limitativ al resurselor hardware pe care le are la dispoziție. Noțiunea de emulare pune în evidență, față de interpretare, un aspect calitativ nou al mașinilor microprogramate: acela de a putea fi și simulatoare sau emulatoare. Acest aspect al microprogramării a apărut din necesitatea compatibilității sistemelor de calcul, mai ales din punct de vedere software. Calculatoarele microprogramate ar putea fi echipate cu memorii de control suplimentare pentru emularea unor calculatoare mai vechi, eliminîndu-se astfel necesitatea reprogramării. Emulatorul va reprezenta, în acest sens, „un set complet de microprograme care, atunci cînd este înscris în memoria de control, definește o mașină"[5]. Mașina pe care se face emularea este numită în general *mașină-gazdă* (host machine), iar mașina care este emulată, *mașină-țintă* (target machine).

Microprogramarea poate deci extinde viața utilă a sistemului, care poate îndeplini noi cerințe prin reprogramarea funcției de control. Acest lucru este adevărat pe timpul întregii vieți a mașinii microprogramate. Posibilitatea de a răspunde unor noi cerințe, precum și posibilitatea de a extinde arhitectura unei mașini pentru a emula o altă mașină sînt avantaje deosebit de importante, îndeosebi pentru calculatoarele deja instalate, unde se pune problema eficientizării conversiei de programe și de date.

2. *Ușurință în dezvoltare și întreținere.* Microprogramarea a fost definită și ca o metodă sistematică de implementare a funcției de control pentru diferite mașini numerice. Aceasta reprezintă un avantaj evident față de metoda *ad-hoc* care caracterizează proiectarea convențională a părților de control. În proiectarea convențională efortul total este limitat în principiu de capacitățile proiectantului. Acesta are responsabilitatea trecerii de la arhitectura sistemului, definită de specificațiile inițiale, la organigramele de timp și de funcționare, precum și responsabilitatea implementării acestor organigrame în hardware. Proiectantul logic are, de asemenea, responsabilitatea proiectării procedurilor de testare și depanare a structurii de control, cea a realizării unei documentații complete. Pe scurt, un proiect de structură de control convențională nu poate fi privit decît ca un întreg care înglobează probleme de proiectare logică propriu-zisă, probleme de testare, întreținere, documentare. Toate aceste probleme sînt dificile, complexe și greu de rezolvat fără erori, mai ales în sistemele mari, tocmai datorită acestei concentrări. Pe de altă parte, microprogramarea, așa cum s-a mai spus, oferă o divizare a proiectului, permițînd o îmbunătățire a calității și vitezei de realizare a produsului. Spre exemplu, în proiectarea unei mașini microprogramate se poate colabora cu un programator cu experiență la proiectarea organigramelor și scrierea microprogramelor. Cu ajutorul unui simulator software sau al unui sistem suport pentru microprogramare se pot verifica, depana și compara diferite soluții pentru a le optimiza din punctul de vedere al vitezei electronice, al volumului de microcod etc. La fel se poate produce și un set clar, complet, standardizat, de documentație. De asemenea, microprogramarea oferă multe posibilități de automatizare a proiectării.

3. *Uniformitate a proiectării.* Legat de avantajul de mai sus, dar suficient de conturat şi important pentru a fi menţionat separat, este şi avantajul uniformităţii caracteristicilor de proiectare. Metoda microprogramării oferă ordine, uniformitate şi modularitate în proiectarea structurilor de control. Datorită acestei uniformităţi, care deseori înseamnă simplitate, se poate aprecia că este nevoie de mai puţine eforturi de învăţare şi experienţă pentru a forma microprogramatori pricepuţi decît proiectanţi de hardware convenţional.

4. *Eforturi de învăţare mai mici.* Datorită abordării simple, metodice şi organizate, microprogramarea este o tehnică de proiectare mai uşor de învăţat. De exemplu, pentru SCM funcţionarea poate fi înţeleasă numai cu ajutorul organigramelor şi microprogramelor scrise simbolic, în timp ce pentru o structură de control realizată convenţional sînt necesare organigrame de timp şi de funcţionare, scheme logice complicate. Microprogramarea s-a dovedit a fi o metodă foarte utilă şi în alte probleme legate de ştiinţa calculatoarelor, cum ar fi cele referitoare la arhitectura şi organizarea sistemelor.

5. *Preţ de cost scăzut.* Un alt motiv pentru care microprogramarea este utilizată extensiv este şi preţul ei scăzut de implementare. Deşi economia de cost este dependentă şi de arhitectura structurii de control, de mijloacele de proiectare disponibile, SCM sînt ieftine în principal datorită dezvoltărilor tehnologice din domeniul memoriilor şi microprocesoarelor. Aceste circuite integrate pe scară largă reprezintă blocuri constructive ieftine cu care se pot implementa uşor şi eficient SCM care să controleze sisteme numerice de diverse complexităţi. Microprogramarea este o tehnică de proiectare care permite utilizarea circuitelor LSI şi deci construirea de structuri de control ieftine, puţin voluminoase, fiabile.

6. *Viteza.* Microprocesoarele *bit-slice*, realizate în tehnologii TTL Schottky sau ECL oferă, împreună cu memoriile semiconductoare foarte rapide, suportul hardware pentru implementarea unor aplicaţii ale microprogramării de foarte mare viteză. Pentru domenii cum ar fi, de exemplu, prelucrarea semnalelor, microprocesoarele MOS, tehnologiile convenţionale, sînt foarte lente şi/sau voluminoase, astfel încît, structurile microprogramate cu microprocesoare *bit-slice* reprezintă (poate cu excepţia unor proiecte cu PLA-uri) cea mai rapidă soluţie realizabilă cu circuite LSI disponibile în comerţ. Deşi tehnologiile bipolare în care se realizează familiile de microprocesoare *bit-slice* nu permit o integrare foarte mare şi deci funcţiile de control se implementează cu un volum mai mare de circuite, SCM realizate cu LSI rămîn încă deosebit de eficiente pentru aplicaţii de mare viteză şi/sau specializate. Astfel, de exemplu, funcţiile unui microprocesor de 16 biţi pot fi implementate pe o singură placă imprimată cu 20—30 circuite integrate, obţinîndu-se o viteză de 15 ori mai mare decît a unui microprocesor realizat în tehnologie NMOS [7].

Un alt punct de vedere asupra vitezei este acela că în calculatoare implementarea microprogramată a unui algoritm este de multe ori mai rapidă decît o implementare, echivalentă funcţional, într-un limbaj maşină.

7.3.2. DEZAVANTAJELE MICROPROGRAMĂRII

1. *Pierdere de performanță.* Acest dezavantaj este principial și se poate datora mai multor motive. În primul rînd, el este un dezavantaj al tuturor structurilor de control programabile care, spre deosebire, de exemplu, de structurile analogice, implementează algoritmii de funcționare secvențial, pas cu pas, numeric. În al doilea rînd, există o pierdere de performanță în SCM în comparație cu structurile realizate convențional sau cu PLA-uri. Datorită puterii limitate a microinstrucțiunii, pot apărea uneori întîrzieri în prelucrare sau în adaptarea SCM la proces. Ceea ce se poate face printr-o logică convențională, pur combinațională, complicată, dar într-o singură perioadă de ceas, se realizează, de cele mai multe ori, în mai multe perioade de ceas, printr-o subrutină de microprogram. În al treilea rînd, rezultă uneori o pierdere de timp, deci de performanță, datorită imposibilității de a suprapune citirea microinstrucțiunii cu execuția ei. Acest dezavantaj poate să apară în cazul SCM de tip *pipeline* numai pentru microinstrucțiuni de test sau permanent la SCM seriale.

2. *Volum, putere consumată mai mari.* La alegerea microprogramării ca metodă de implementare a unei structuri de control trebuie ținut cont și de faptul că SCM realizate cu circuite LSI bipolare sînt mai voluminoase și consumă mai multă putere în comparație cu structurile de control realizate cu circuite integrate LSI MOS. Aceasta ca o particularizare a faptului că tehnologiile care oferă viteze mai ridicate (TTL Schottky, ECL) implică o putere consumată mai mare și un grad de integrare mai mic.

7.4. APLICAȚII ALE MICROPROGRAMĂRII

Așa cum s-a mai spus, microprogramarea a întîrziat mult în trecerea de la concepție, elaborare teoretică, la utilizarea extensivă. O tehnologie incomodă, greu de utilizat, scumpă, o lipsă de experimentări și cercetări practice, neincluderea metodei în programele de învățămînt, inerție, lipsa de tradiție, toate sînt cauze ale acestei situații. Odată depășite inconvenientele microprogramarea va fi înconjurată de o aură de pragmatism, iar după apariția minicalculatoarelor microprogramabile la utilizator, a microprocesoarelor *bit-slice*, a memoriilor de control LSI, practicienii au la dispoziție o bază voluminoasă și ieftină pentru a investiga noi aplicații.

Deși aplicațiile microprogramării sînt încă într-o perioadă de început, de dezvoltare, ele se pot clasifica în trei categorii mari:

— proiectarea și implementarea de *calculatoare universale*;
— dezvoltarea de *dispozitive de control*;
— dezvoltarea de *dispozitive specializate*.

În domeniul calculatoarelor universale aplicațiile se pot grupa, la rîndul lor, conform următoarelor aspecte mai importante:

— implementarea repertoriilor standard de instrucțiuni și/sau adăugarea de instrucțiuni suplimentare;

— sisteme de operare implementate prin microprograme speciale (firmware de bază);
— emularea altor calculatoare;
— implementarea limbajelor de nivel înalt.

Implementarea microprogramată a repertoriului de instrucțiuni înseamnă de fapt realizarea unor unități centrale microprogramate sau, mai general, microprogramabile. În acest caz, prin firmware se asigură citirea instrucțiunilor din memoria operativă, interpretarea și executarea lor beneficiind de toate avantajele microprogramării. În plus, calculatoarele universale dispun, în general, de coduri de operație neutilizate. Acestea pot fi folosite pentru introducerea de noi instrucțiuni. Structurile de control microprogramabile permit ca lucrul să fie relativ ușor de realizat, noile coduri reprezentînd puncte de intrare ale unor noi microrutine. Dintre instrucțiunile suplimentare care se pot implementa în acest fel menționăm [8] pe cele care se referă la:

— operații aritmetice în virgulă mobilă;
— operații aritmetice cu numere reprezentate în cod BCD;
— manipulări de blocuri de date;
— calculul codurilor corectoare de erori;
— operații cu stive;
— rutine de I/E;
— aproximări și interpolări de curbe;
— operații cu tabele;
— operații cu caractere și șiruri de caractere;
— asigurarea protocolului de comunicație între sisteme;
— filtrare, analiză spectrală, prelucrare de semnale;
— recunoașterea formelor;
— prelucrarea imaginilor.

Posibilitatea de a introduce instrucțiuni de acest tip într-un calculator universal îi mărește considerabil eficiența și îi prelungește timpul de viață utilă. Avînd la dispoziție o unitate centrală ușor microprogramabilă, introducerea unei noi instrucțiuni este foarte comodă și înseamnă: stabilirea algoritmului, scrierea microrutinei, analiza timpului de execuție.

În domeniul utilizării microprogramării în sistemele de operare, aplicațiile au urmat două direcții principale:

— implementarea de primitive microprogramate utilizate permanent de sistemul de operare și
— implementarea de porțiuni importante din sistemele de operare, parțial sau în întregime, prin microcod.

Primitivele care pot fi implementate prin microprogram sînt acelea des apelate și/sau a căror execuție este prea lentă prin software. Printre acestea se pot menționa primitivele de sincronizare a programelor, primitivele de manipulare a cozilor de așteptare, cele care comută starea program.

Alte microprograme de bază a căror execuție se poate face în mod continuu sau la comandă, efectuînd funcțiuni independente de software-ul de bază sau legate indirect de el, se referă la:

— explorarea, eșantionarea și introducerea datelor analogice;
— operații cu matrice;
— conversii de coordonate;
— funcții cu caracter statistic;
— unele operații de intrare/ieșire.

Dînd o definiție cuprinzătoare a mașinilor-țintă, cele mai multe aplicații din domeniul calculatoarelor universale pot fi considerate ca aplicații de tip *emulare*. Harry Katzan, Jr. consideră chiar că „emularea este principala aplicație a microprogramării" [9]. Pentru a se putea aprecia posibilitățile de emulare și eficiența ei trebuie cunoscute bine mașina-țintă, mașina-gazdă, precum și unele tehnici de microprogramare specifice acestui tip de aplicații. Printre aceste tehnici specifice emulării se pot enumera cele referitoare la utilizarea registrelor de bază, a tabelelor, la întrebuințarea efectivă a operațiilor cu memoria principală. În general, toate aceste tehnici sînt independente de mașină. Un microprogram emulator ar putea fi compus dintr-un tabel de salturi la microrutine, o microrutină de extragere (fetch) și microrutinele de execuție pentru fiecare tip de instrucțiune emulată. Proiectarea unui emulator presupune, de asemenea, să se țină cont și de factori cum sînt filozofia de funcționare a calculatorului, formatul instrucțiunii.

Dezvoltarea microprogramelor emulatoare a fost deseori împiedicată de lipsa unor facilități corespunzătoare. Chiar atunci cînd au existat asemenea facilități cum sînt sistemele suport pentru microprogramare, sau simulatoarele, s-au ridicat unele probleme legate de lipsa unui context real de funcționare și de utilizarea eficientă a timpului-mașină. În zilele noastre s-au dezvoltat sisteme complexe care pot asigura:

— mai multe calculatoare-țintă rezidente implementate prin emulare;
— controlul interactiv de către utilizator al acestor emulări;
— intrare/ieșire din emulări pe dispozitive periferice reale;
— depanare simbolică atît a microcodului emulator, cît și a codului mașină emulat.

Ca exemplu se poate da sistemul PRIM care este un sistem emulator accesibil unui mare număr de utilizatori, prin intermediul rețelei de calculatoare ARPANET.

În afară de asigurarea compatibilității între calculatoare, emularea servește și în scopuri de cercetare, facilitînd analiza unor idei noi cum ar fi mașinile ternare, mașinile care lucrează cu cîmpuri Galois sau calculatoarele specializate pentru baze de date.

Implementarea *limbajelor de nivel înalt*, cum sînt FORTRAN, ALGOL, PL/1, LISP, BASIC, PASCAL, pe calculatoare prevăzute cu un repertoriu de instrucțiuni obișnuit, nu este atît de simplă și întîmpină destule greutăți. În lipsa unor instrucțiuni adecvate scrierea compilatoarelor pentru limbaje de nivel înalt este mai laborioasă și duce la ocuparea unui spațiu mai mare de memorie. Adaptarea, îmbogățirea repertoriului de instrucțiuni prin utilizarea de microrutine care să interpreteze instrucțiunile limbajului de nivel

înalt, va duce la o mărire a vitezei de compilare și la reducerea spațiului de memorie ocupat de compilator. Aceste optimizări implică măsurători de performanță ale unui program aflat în execuție, analiza dinamică a informației procesate pentru identificarea deficiențelor de arhitectură și, în final, sinteza de noi instrucțiuni (microrutine) care să remedieze aceste deficiențe. În scopul implementării limbajelor de nivel înalt s-au elaborat tehnici speciale cum sînt cea de „migrare verticală" sau cea de „redefinire dinamică" a arhitecturii calculatorului. Migrarea verticală este o tehnică ce îmbunătățește performanțele sistemului prin deplasarea, migrarea, primitivelor software în „straturile" unui sistem hardware/firmware/software structurat ierarhic [10]. În metoda redefinirii dinamice a arhitecturii se utilizează un compilator ce generează pentru fiecare program în limbaj de nivel înalt un microprogram care definește arhitectura unei mașini-țintă optime pentru rularea eficientă a programului compilat. În faza următoare, compilatorul va genera un program în „limbaj mașină", specific noii arhitecturi definite anterior, reprezentînd programul în limbaj de nivel înalt compilat.

Aplicațiile din domeniul *dispozitivelor de control* se referă la diversele dispozitive dintr-un sistem de calcul funcționînd în esență ca mici structuri de calcul. Este vorba aici, în principal, despre canalele de I/E și cuploarele pentru periferice. Aceste dispozitive pot fi concepute ca procesoare specializate, cu un număr mai mic de registre și fără memorie principală. În sistemele de calcul moderne dispozitivele de control al I/E, cum sînt canalele și cuploarele, sînt frecvent implementate ca structuri de control microprogramate. Utilizarea microprogramării conduce aici la realizarea unor structuri de control flexibile, dezvoltabile, care se pretează atît pentru sistemele de calcul mari cu periferie complexă, de mare viteză, cît și în sisteme de calcul mai simple. Realizarea de unități de legătură și canale de intrare/ieșire microprogramate conduce la obținerea unor structuri standardizate, la organizarea mai bună a proiectării părții de I/E din sistemele de calcul, la compactizare și mărirea fiabilității. Problemele deosebite care se pun în asemenea proiecte sînt legate în special de viteza de lucru. După cum se știe, canalele și cuploarele pentru periferice sînt, mai ales în sistemele de calcul mari, dispozitive care interconectează unități funcționale cu interfețe complicate atît ca număr de semnale, cît și ca protocol de interfațare. Aceste dispozitive prelucrează simplu informația dar, în schimb, trebuie să fie specializate mai mult pe manevrarea registrelor, testare de indicatori, secvențiere. Deoarece dispozitivele de I/E este necesar să asigure rate de transfer al informației mari, impuse în primul rînd de viteza perifericelor cuplate, canalele și cuploarele vor fi în general mașini microprogramate orizontal. Controlul prin microprogram asigură, de asemenea, dispozitivelor de I/E o manevrare mai ușoară a semnalelor de interfață, o implementare mai flexibilă a algoritmilor de funcționare, o adaptabilitate relativ comodă la noi echipamente periferice sau la schimbarea modului de lucru.

Aplicațiile din domeniul *dispozitivelor specializate* se referă la controlul proceselor fizice, la prelucrarea de semnale, la monitorizarea funcționării unor

diverse aparate, în general la toate aplicațiile din afara domeniului propriu-zis al calculatoarelor care impun cel puțin una din următoarele restricții:

— viteză de lucru foarte mare;

— control foarte specializat.

Utilizarea microprocesoarelor *bit-slice* microprogramabile conduce la realizarea de SCM care să prezinte ambele caracteristici. Într-adevăr, aceste microprocesoare sînt blocuri electronice de mare viteză ce permit implementarea unor structuri de control care să realizeze un set limitat de funcțiuni adaptate la aplicație, asigurînd în același timp, prin microprogramare, flexibilitatea, modularitatea, compactitatea caracteristice tuturor structurilor cu microprocesoare.

Domeniul dispozitivelor specializate cuprinde aplicații ale microprocesoarelor *bit-slice* în realizarea de procesoare numerice de semnale a căror gamă variază în complexitate de la procesoare mici, specializate pentru implementarea filtrelor numerice, pînă la mașini mari, programabile, orientate pe transformate Fourier rapide (Fast Fourier Transform-FFT). De asemenea, acest domeniu cuprinde aplicații de timp real pentru controlul unor procese de mare viteză, aplicații din domeniul comunicațiilor cum ar fi, de exemplu, modemuri, sau așa-numitele canale inteligente pentru prelucrarea de mesaje, aplicații legate de procesarea informației radar, de pildă pentru controlul traficului aerian și aplicații legate de atît de vasta problemă a prelucrării imaginilor.

În acest capitol s-au trecut succint în revistă conceptele și principiile mai importante care stau la baza microprogramării ca metodă modernă de proiectare a unor structuri de control numerice, avantajele și dezavantajele acestei tehnici de proiectare, principalele domenii de aplicare a ei. În capitolele următoare ne vom ocupa de probleme referitoare la dezvoltarea SCM, prezentînd unele mijloace suport pentru microprogramare; vom descrie cele mai importante familii de microprocesoare *bit-slice*, împreună cu unele circuite auxiliare necesare implementării de structuri microprogramate. Vom prezenta, de asemenea, unele aplicații mai semnificative care să ilustreze utilizarea metodei, particularitățile de folosire a microprocesoarelor *bit-slice*. Încercăm astfel să oferim proiectanților de structuri de control numerice majoritatea elementelor, metodologice și tehnologice, care stau la baza proiectării structurilor de control microprogramate.

BIBLIOGRAFIE

1. WILKES, M.V., *The Best Way to Design an Automatic Calculating Machine*, Report of the Manchester University Computer Inaugural Conference, Electrical Engineering Department of Manchester Universi'ty, Manchester, Anglia, Iulie 1951, republicat în „Computer Design Development-Principal Papers", editor Earl E. Swartzlander, Jr., Hayden Book Co., Rochelle Park, New Jersey, 1976, menționat în [2].
2. RAUSHER, T.G.; ADAMS, P.M., *Microprogramming: A Tutorial and Survey of Recent Development*, IEEE Transactions on Computers, 1980, C-29, 1, p. 2—20.
3. SALISBURRY, A.B., *Microprogrammable Computer Architecture*, Elsevier, New York, 1976.
4. HUSSON, S.S., *Microprogramming. Principles and Practices*, Prentice-Hall, Englewood Cliffs, New Jersey, 1970.

5. ROSIN, R.F., *Contemporary Concepts of Microprogramming and Emulation*, Computer Surveys, Vol. 1. Decembrie 1969, menționat în [2].
6. DAVIDSON, S.; SHRIVER, B.D., *An overview of Firmware Engineering*, Computer, 1978, 11, 5, p. 21—33.
7. DANCEA, I., *Microprocesoare, arhitectură internă, programare, aplicații*, Editura Dacia, Cluj-Napoca, 1979.
8. PETRESCU, A., *Microprogramare. Principii și aplicații*, Editura Tehnică, București, 1975.
9. KATZAN, Jr., H., *Microprogramming Primer*, McGraw-Hill Book Company, New York, 1977, p. 103—118, 197—223.
10. STOCKENBERG, J.; DAM, ANDRIES VAN, *Vertical Migration for Performance Enhancement in Layered Hardware/Firmware/Software Systems*, Computer, 1978, 11, 5, p. 35—50.
11. BAER, J.L.; KOYAMA, B., *On the Minimization of the Width of the Control Memory of Microprogrammed Processors*, IEEE Transactions on Computers, 1979, C-28, 4, p. 310—316.
12. LUCIDO, T.P.; CHATTERGY, R.; POOCH, U.W., *A survey of Microprogram Verification and Validation Methods*, The Computer Journal 1981, 24, 2, p. 139—142.
13. ROBERTSON, E.L., *Microcode Bit Optimization is NP-Complete*, IEEE Transactions on Computers, 1979, C-28, 4, p. 316—319.
14. ȚĂPUȘ, N., *Contribuții la elaborarea de noi structuri de sisteme de calcul microprogramate*, Teză de doctorat, Institutul Politehnic București, Facultatea Automatică, București, 1981.

SISTEME SUPORT PENTRU MICROPROGRAMARE

8.1. UTILITATEA SISTEMELOR SUPORT PENTRU MICROPROGRAMARE

8.1.1. FUNCŢIILE SISTEMELOR SUPORT PENTRU MICROPROGRAMARE

Sistemele suport pentru dezvoltarea structurilor de control microprogramate încep să apară şi să-şi impună utilitatea mai ales după lansarea comercială a microprocesoarelor *bit-slice*, odată cu creşterea şi diversificarea aplicaţiilor microprogramării.

Deoarece microprogramele care conţin informaţia pentru controlul resurselor hardware ale maşinii sînt păstrate într-o memorie şi sînt executate ca programe stocate, proiectarea SCM va avea şi un aspect software. De aceea termenul *firmware*, introdus de *Asher Opler*, care desemnează microprogramele rezidente în memoria de control, pare cel mai potrivit. Acest aspect software permite divizarea proiectului unei maşini microprogramate într-un proiect de hardware şi un proiect de software (v. cap. 2), asigurîndu-se astfel un bun paralelism în proiectare. Specificaţiile generale ale sistemului conduc întîi la un proiect hardware global care defineşte, printre altele, intrarea/ /ieşirea, structura generală a resurselor hardware, microinstrucţiunea, microoperaţiile maşinii. Acestea constituie punctele de referinţă utilizate de inginerul proiectant (pentru a realiza hardware-ul) şi de microprogramator (cînd complexitatea SCM impune persoane specializate pentru scrierea microprogramelor) în vederea elaborării microcodului. Cele două căi de dezvoltare paralele încep să se întrepătrundă la punerea la punct, atunci cînd prototipul va trebui să execute părţi de microprogram. Etapa de punere la punct este iterativă, ea derulîndu-se în mai multe treceri, fiecare cu schimbări sau corectări de microprogram şi/sau modificări de hardware.

Dezvoltarea maşinilor microprogramate, necesitînd eforturi de proiectare hardware şi software interdependente, sinergistice, optimizarea proiectării unor astfel de maşini a condus la apariţia sistemelor suport pentru microprogramare, SSM. Aceste sisteme au trei funcţii principale.

Prima funcţie este aceea de a asigura o memorie de tip WCS care să servească drept memorie de microprograme. Această memorie va trebui să aibă un timp de acces foarte mic, pentru a putea fi utilizată în condiţiile reale de funcţionare ale maşinii microprogramate care se pune la punct. De asemenea, memoria WCS trebuie să fie organizată cît mai flexibil pentru a putea fi reconfigurată comod conform unor diverse structuri ale memoriei de microprograme.

A doua funcţie principală a SSM este aceea de a asigura mijloace de diagnosticare hardware pentru maşina microprogramată care se află în curs de punere la punct. Ne referim aici la posibilităţi de afişare şi/sau modificare

a microinstrucțiunii și a stărilor principalelor resurse hardware ale mașinii microprogramate, la posibilități de generare și modificare a ciclurilor de microinstrucțiune, la posibilitatea de a lucra în mod automat sau în mod pas-cu-pas, cu pornire și/sau oprire la adrese prestabilite.

A treia funcție a SSM este să asigure software-ul pentru dezvoltarea microcodului. Includem aici programe de tip monitor, asamblor de microprograme, meta-asamblor, editor de text, compilatoare pentru limbaje de nivel înalt. Cu ajutorul monitorului se poate scrie sau altera conținutul memoriei de microprograme, se poate muta o zonă de memorie peste alta, se poate introduce sau extrage un microprogram de pe/pe un suport extern. Asamblorul, un program dedicat, permite scrierea simbolică a microprogramelor conform unei anumite structuri de microinstrucțiune, obținerea codului-obiect și a benzilor perforate necesare programării memoriilor PROM. Meta-asambloarele lucrează în două faze distincte. În prima fază se definesc formatul microinstrucțiunii și codurile mnemonice ale tuturor microoperațiilor, stabilindu-se un limbaj de asamblare. În faza următoare, de asamblare propriu-zisă, microinstrucțiunile scrise în limbajul definit anterior sînt translatate în cod-obiect. Ca și în cazul asambloarelor convenționale, un meta-asamblor poate avea și o fază de postprocesare în care se obține din codul-obiect, generat în faza de asamblare, un cod-obiect reorganizat ce poate fi încărcat direct în memoria WCS a sistemului de dezvoltare sau un cod obiect care poate fi ars în memorii PROM sau în PLA-uri. Editorul de text permite — pe parcursul punerii la punct — efectuarea corecțiilor într-un microprogram-sursă. Limbajele de nivel înalt pentru microprogramare, limbaje în general independente de mașină, oferă posibilitatea scrierii optime a microprogramelor, rezolvînd probleme complexe cum sînt utilizarea resurselor hardware, reducerea timpului de execuție al microprogramului, optimizarea memoriei de microprograme [1].

Sistemele suport pentru microprogramare pot fi concepute ca unități independente [2] sau ca opțiuni ale unor sisteme de dezvoltare pentru microprocesoare *single-chip* [3, 4, 5, 6, 7]. Considerăm că a doua abordare este mai bună avînd în vedere că un sistem suport de dezvoltare este cu atît mai util cu cît este folosit pentru proiectarea unei game cît mai largi de aplicații. De asemenea, considerăm ca utilă elaborarea unor sisteme de dezvoltare compatibile cu sisteme de calculatoare universale ca FELIX C-256, I-100, FELIX M-18, ceea ce ar permite utilizarea unui set bogat de resurse hardware și software.

8.1.2. *SYSTEM* 29 — UN SISTEM SUPORT PENTRU MICROPROGRAMARE

Pentru o mai bună înțelegere a funcțiilor și utilității unui SSM vom descrie în continuare, pe scurt, sistemul denumit *System* 29, realizat de firma AMD [4].

Structura *System*-ului 29 este dată în figura 8.1. După cum se vede, sistemul se împarte în două secțiuni: microcalculatorul *host* (gazdă) și sistemul suport propriu-zis pentru microprogramare. Microcalculatorul *host* este construit cu un microprocesor 8080 și are o memorie RAM de 32 Kocteți exten-

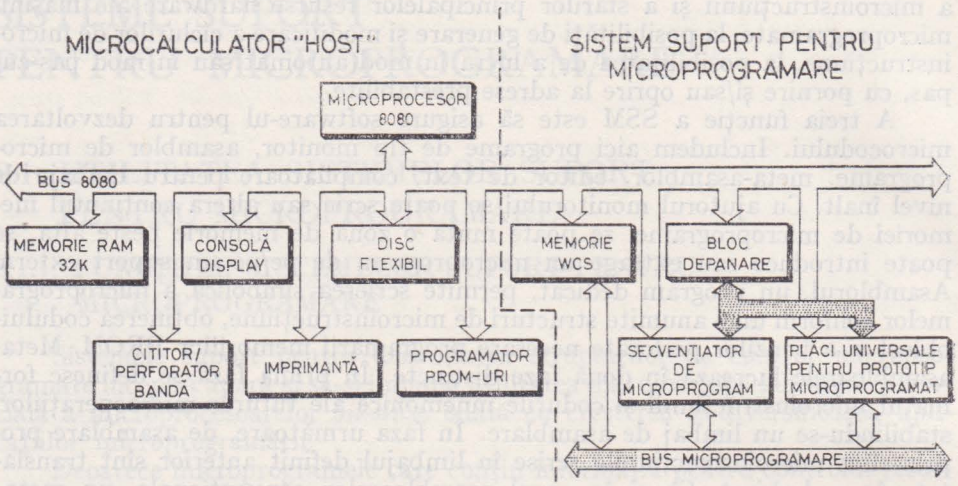


Fig. 8.1. Structura sistemului suport pentru microprogramare *System 29*

sibilă pînă la 64 Kocteți. Poate fi cuplat opțional cu diverse dispozitive periferice ca disc flexibil, consolă-display, cititor/perforator de bandă, imprimantă, prorgamator de PROM-uri etc. Microcalculatorul *gazdă* controlează, de asemenea, și sistemul suport pentru microprogramare.

SSM se interfațează cu mașina microprogramată care se pune la punct. Această mașină poate fi construită pe plăcile universale care sînt prevăzute în SSM sau, în afara sistemului, într-o tehnologie a utilizatorului. Secțiunea SSM mai cuprinde memoria WCS, blocul pentru depanare, diverse blocuri destinate aplicațiilor.

Memoria WCS a *System*-ului 29 este expandabilă și reconfigurabilă prin adăugarea de module standard. Ea poate fi folosită în configurații de la 1k ×64 biți la 4k ×64 biți sau 4k ×128 biți. Sistemul este prevăzut cu module standard de 1k ×64 biți de viteze diverse. Astfel, memoria WCS poate fi folosită în aplicații de mare viteză în care timpul de acces la memoria de microprograme este de maximum 50 ns, în configurații de la 1k ×64 biți la 4k ×64 biți sau la 2k ×128 biți. Pentru aplicații de viteză medie memoria WCS poate fi reconfigurată în gama de la 2k ×64 biți la 4k ×128 biți.

Blocul destinat depanării realizează funcții de control al ceasului și al adresei de microprogram. Astfel, ceasul mașinii microprogramate poate fi controlat de la panou, de la consola-display sau direct de către mașina microprogramată care se pune la punct și poate fi utilizat în mai multe moduri: oprit, automat, un pas (un ciclu de ceas), un micropas (un ciclu de microinstrucțiune), un anumit număr de micropași. Blocul pentru depanare permite utilizarea de întreruperi, *breakpoint*-uri, sau cuvinte de sincronizare. De asemenea, microprogramul se poate lansa de la o adresă selectată, sau se poate branșa la o anumită adresă, într-un anumit ciclu de microinstrucțiune. Blocul permite memorarea ultimelor 256 adrese de microprogram executate și afișarea lor pe display, la oprirea ceasului. *System*-ul 29 asigură, prin intermediul acestui bloc, definirea și monitorizarea a maximum 64 puncte de test

din mașina care se pune la punct. Aceste puncte, ce pot constitui, de exemplu, adresa de microprogram, microinstrucțiunea, ieșirea UAL, alte *bus*-uri interne și resurse ale mașinii microprogramate în curs de punere la punct, sînt memorate cu o frecvență selectabilă de utilizator. Pe parcursul punerii la punct utilizatorul poate opri ceasul și interoga starea acestor puncte de test.

Blocurile pentru aplicații sînt blocuri constructive care pot intra în alcătuirea unei mașini microprogramate realizate cu circuite din familia Am 2900. Ca exemplu se poate da un secvențiator de microprogram realizat cu circuite Am 2911.

Resursele software ale *System*-ului 29 cuprind un sistem de operare pe disc, software pentru generarea microprogramelor, software suport pentru microprograme, software 8080.

Sistemul de operare pe disc, AMDOS 29, realizează gestiunea fișierelor de pe disc și cuprinde toate rutinele de I/E necesare utilizării dispozitivelor periferice. Pe fiecare disc se pot crea 64 de fișiere distincte de maximum 240 Koct.

Software-ul pentru generarea microprogramelor cuprinde următoarele:

AMDASM 29 — asamblor de microprograme care permite definirea formatului microinstrucțiunii și asamblarea microprogramelor scrise în limbajul definit de utilizator;

AMSCRM 29 — program care reorganizează microcodul conform unei structuri dorite pentru memoria de control;

AMPROM 29 — program care generează microcodul-obiect într-o formă acceptată de memoria de control;

AMMAP 29 — program care generează adresele de secvențe ce se încarcă în memoria RAM pentru *mapare* din blocul secvențiator.

Software-ul suport pentru microprograme cuprinde programe care încarcă, salvează și depanează microcodul pe parcursul punerii la punct firmware/hardware:

LBPM — program care încarcă de pe disc memoria WCS sau memoria de *mapare* din blocul secvențiator;

VBPM — program care verifică memoria WCS sau memoria de *mapare* prin comparare cu informația păstrată pe disc;

SBPM — salvează pe disc microprogramul din WCS sau conținutul memoriei de *mapare*;

RBPM — restaurează microprogramul salvat cu SBPM;

DDT29 — program pentru depanare dinamică, prin intermediul unor comenzi de tipul TRACE, STEP, HALT, RUN, DISPLAY, MODIFY și JUMP, oferă un control dinamic puternic al mașinii microprogramate.

Software-ul 8080 permite scrierea de programe pentru microcalculatorul *host*, acestea adaptînd *System*-ul 29 la diverse genuri de aplicații. În această categorie de programe intră:

ASM — asamblor 8080;

DDT — program pentru depanare dinamică, permite urmărirea simbolică a execuției programelor, depanarea, testarea, dezasamblarea;

LOAD — program care pregătește pentru execuția directă un fișier-memorie dintr-un fișier asamblat.

Utilizarea *System*-ului 29 este foarte simplă. După inițializare, sistemul de operare AMDOS 29 este încărcat automat de pe disc în memoria RAM permițînd începerea lucrului. Prin scrierea de programe simple sistemul poate servi în continuare la analiza, proiectarea și depanarea structurii hardware a mașinii microprogramate *target*.

După stabilirea formatului microinstrucțiunii, se utilizează microasamblorul AMDASM 29 pentru scrierea fișierului de definiție, a fișierelor-sursă scrise în limbaj mnemonic și a fișierelor-obiect. După asamblare fișierele-obiect sînt transferate pe disc. Integrarea firmware/hardware se face prin încărcarea microcodului în memoria WCS și utilizarea corespunzătoare a resurselor *System*-ului 29, executîndu-se operații cum ar fi:

— verificarea și editarea microcodului din WCS;
— verificarea în pas cu pas și automat a microrutinelor;
— *breakpoint*-uri sau opriri pe secvențe de adrese;
— salturi la anumite adrese în anumiți cicli de microinstrucțiune;
— utilizarea de microrutine de diagnosticare special construite și care pot să nu facă parte din microcodul final;
— monitorizarea în timp real a maximum 64 puncte de test.

După punerea la punct a mașinii microprogramate urmează o fază de postprocesare în care, cu ajutorul programului AMPROM 29, microcodul este reorganizat pentru a se genera benzile perforate pentru programatorul de

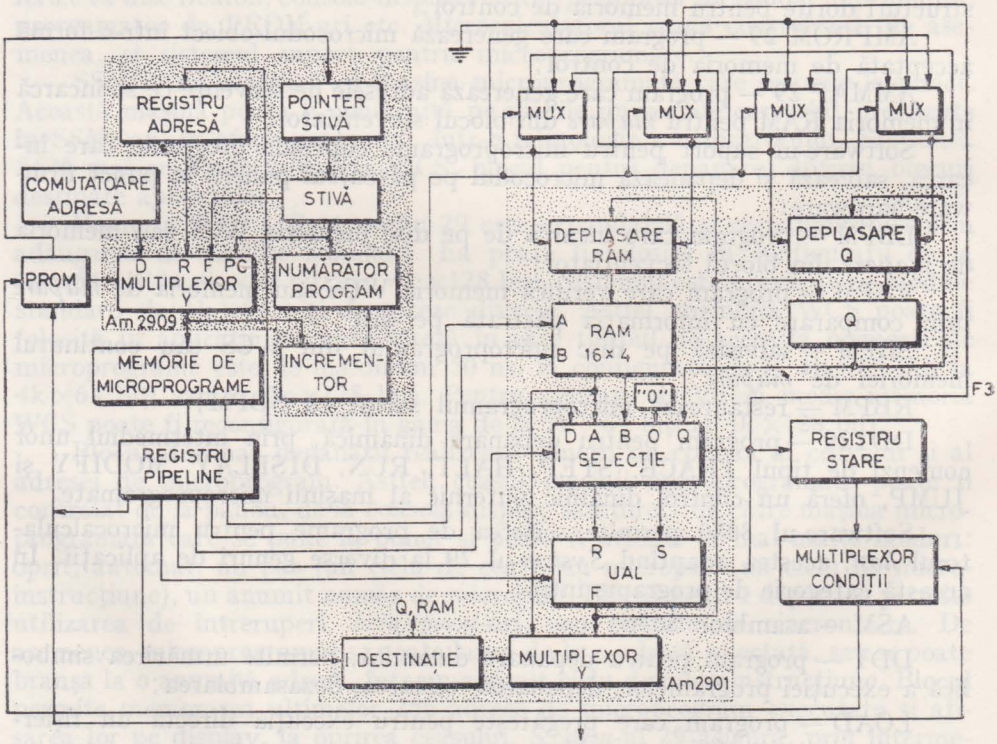


Fig. 8.2. O unitate centrală pe 4 biți

PROM-uri sau, dacă este cazul, pentru a acționa direct programatorul cuplat la *System*-ul 29. Mai departe, cu ajutorul programului AMSCRM, microinstrucțiunea se poate și ea reorganiza, înainte de listare, conform structurii memoriei de control.

8.1.3. UN EXEMPLU DE UTILIZARE A *SYSTEM*-ULUI 29

Posibilitățile unui sistem suport pentru microprogramare, în speță *System*-ul 29, pot fi puse în evidență prin microprogramarea unei mașini simple cum este cea a cărei structură se dă în figura 8.2.

Se observă că mașina microprogramată din figura 8.2, o unitate centrală de calculator pe 4 biți, este alcătuită dintr-un secvențiator de microprogram Am 2909, o memorie de microprograme, un registru *pipeline* pentru microinstrucțiune, un PROM pentru controlul secvențiatorului Am 2909, un microprocesor *bit-slice* Am 2901, multiplexoare pentru deplasări, un registru de stare și un multiplexor pentru selecția condiției testate.

Microinstrucțiunea de control are 32 biți organizați în 11 cîmpuri. Formatul microinstrucțiunii este dat în figura 8.3.

În continuare, cu ajutorul microasamblorului AMDASM 29, se vor defini cîmpurile și codurile mnemonice, utilizate apoi pentru microprogramarea mașinii din figura 8.2.

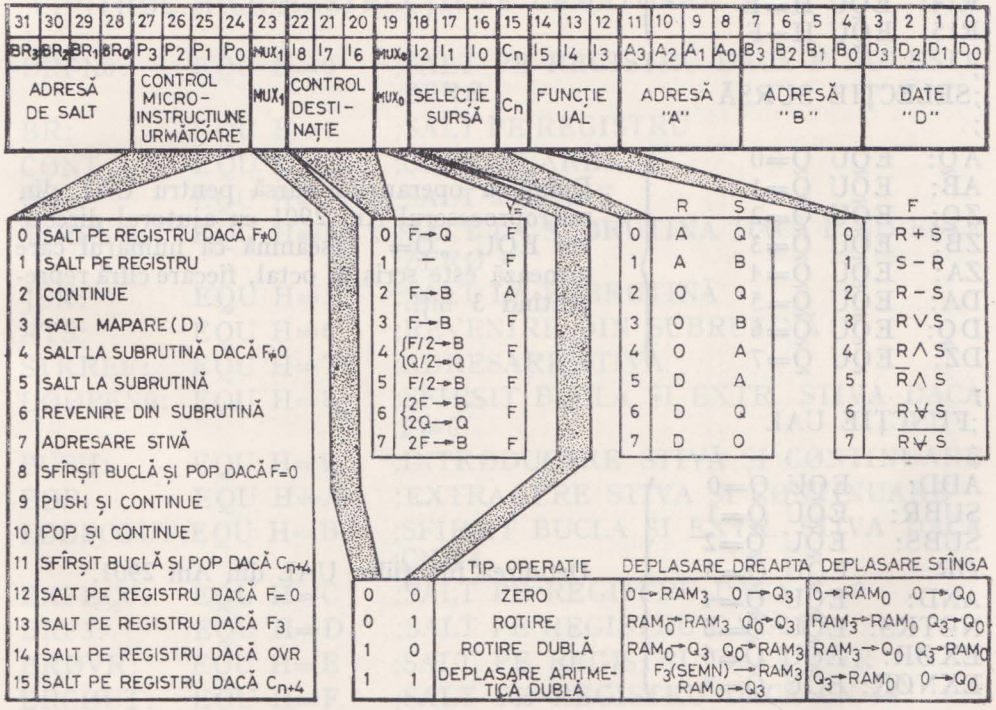


Fig. 8.3. Microinstrucțiunea de control pentru unitatea centrală din figura 8.2.

```
TITLE EXEMPLU DE UTILIZARE A SISTEMULUI 29
;DEFINIȚII
;
WØRD 32
;
;REGISTRE DE LUCRU
;
R0:   EQU H=0
R1:   EQU H=1
R2:   EQU H=2
R3:   EQU H=3
R4:   EQU H=4
R5:   EQU H=5
R6:   EQU H=6
R7:   EQU H=7
R8:   EQU H=8
R9:   EQU H=9
R10:  EQU H=A
R11:  EQU H=B
R12:  EQU H=C
R13:  EQU H=D
R14:  EQU H=E
R15:  EQU H=F
;
;SELECȚIE SURSĂ
;
AQ:   EQU Q=0
AB:   EQU Q=1
ZQ:   EQU Q=2
ZB:   EQU Q=3
ZA:   EQU Q=4
DA:   EQU Q=5
DQ:   EQU Q=6
DZ:   EQU Q=7
;
;FUNCȚIE UAL
;
ADD:    EQU Q=0
SUBR:   EQU Q=1
SUBS:   EQU Q=2
ØR:     EQU Q=3
AND:    EQU Q=4
NØTRS:  EQU Q=5
EXØR:   EQU Q=6
EXNØR:  EQU Q=7
;
```

Definirea registrelor de lucru din memoria RAM a microprocesorului Am 2901 cu ajutorul directivei EQU. „H=" înseamnă că numărul care urmează este scris în hexazecimal, fiecare cifră reprezentînd 4 biți.

Definirea operanzilor-sursă pentru UAL din microprocesorul Am 2901 cu ajutorul directivei EQU. „Q=" înseamnă că numărul care urmează este scris în octal, fiecare cifră reprezentînd 3 biți.

Definirea funcțiilor UAL din Am 2901.

```
;CØNTRØL DESTINAȚIE
;
QREG:   EQU Q=0
NØP:    EQU Q=1
RAMA:   EQU Q=2
RAMF:   EQU Q=3
RAMQD:  EQU Q=4
RAMD:   EQU Q=5
RAMQU:  EQU Q=6
RAMU:   EQU Q=7
;
;CØNTRØL DEPLASĂRI
;
SHIFT:  DEF 8X,B=0,3X,B=0,19X
RØTATE: DEF 8X,B=0,3X,B=1,19X
DBLRØT: DEF 8X,B=1,3X,B=0,19X
ARITH:  DEF 8X,B=1,3X,B=1,19X
;
;CØNTRØL MICRØINSTRUCȚIUNE URMĂTØARE
;
BRFN0:      EQU H=0   ;SALT PE REGISTRU DACĂ F NU ESTE
                      ;ZERØ
BR:         EQU H=1   ;SALT PE REGISTRU
CØNT:       EQU H=2   ;CØNTINUARE
BM:         EQU H=3   ;SALT MAPARE
JSRFNO:     EQU H=4   ;SALT LA SUBRUTINA DACĂ F NU ESTE
                      ;ZERØ
JSR:        EQU H=5   ;SALT LA SUBRUTINĂ
RTS:        EQU H=6   ;REVENIRE DIN SUBRUTINĂ
STKREF:     EQU H=7   ;ADRESARE STIVA
LØØPFN0:    EQU H=8   ;SFIRSIT BUCLA SI EXTR. STIVA DACA
                      ;F=0
PUSH:       EQU H=9   ;INTRØDUCERE STIVĂ ȘI CØNTINUARE
PØP:        EQU H=A   ;EXTRAGERE STIVA ȘI CØNTINUARE
LØØPCØUT:   EQU H=B   ;SFIRSIT BUCLA ȘI EXTR. STIVA DACA
                      ;CN+4
BRFEQ0:     EQU H=C   ;SALT PE REGISTRU DACĂ F=0
BRF3:       EQU H=D   ;SALT PE REGISTRU DACĂ F3
BRØVR:      EQU H=E   ;SALT PE REGISTRU DACĂ ØVR
BRCØUT:     EQU H=F   ;SALT PE REGISTRU DACĂ CN+4
;
```

Definirea cîmpului de control al destinației UAL din Am 2901.

Definirea cîmpurilor MUX1 și MUX0 care controlează deplasările cu ajutorul instrucțiunii DEF. X semnifică biți fără importanță. „B=" înseamnă că numărul care urmează este scris în binar, fiecare cifră reprezentînd un bit.

```
;DIVERSE
;
CN0:   EQU B=0
CN1:   EQU B=1
LØW:   EQU B=0
HIGH:  EQU B=1
ZERØ:  EQU B=0
ØNE:   EQU B=1
;
;CØNTRØL AM2901
;
AM2901: DEF 9X,3VQ=1,1X,3VX,1VX,3VX,
                    4VX,4VX,4X
;
;CØNTRØL AM2909
;
AM2909: DEF 4VX,4VH=2,24X
;
;INTRAREA D IN AM2901
;
DIN: DEF 28X,4VH=0
;
END
```

Definiții de format care conțin cîmpuri fără importanță (X) și cîmpuri variabile (V). Fiecare cîmp variabil poate avea o valoare în lipsă, folosită de AMDASM atunci cînd cîmpul nu este specificat în microinstrucțiune.

După definirea codurilor mnemonice, care alcătuiesc de fapt un limbaj de asamblare, proiectantul poate trece la scrierea microprogramelor de control pentru mașina din figura 8.2.

Ca exemplu vom scrie o secvență care numără biții „1" din trei cuvinte succesive V0, V1 și V2 [4]. O astfel de secvență poate fi utilizată pentru calculul parității sau în unele proceduri de comunicație. Organigrama acestei secvențe este dată în figura 8.4. Microcodul corespunzător, scris în AMDASM 29, este dat mai jos.

```
 1          TITLE EXEMPLU DE UTILIZARE A SISTEMULUI 29
 2          ØRG 0
 3          AM2909&AM2901 RAMF,DZ,,ØR,,R0&DIN H=F
 4          AM2909&AM2901 RAMF,DZ,,ØR,,R1&DIN 9
 5          AM2909&AM2901 RAMF,DZ,,ØR,,R2&DIN 0
 6          AM2909&AM2901 RAMF,ZB,,AND,,R3
 7          AM2909&AM2901 RAMF,DZ,,ØR,,R4&DIN 4
 8 ADR8:    AM2909&AM2901          ,DA,,AND,R0,R0&DIN 1
 9          AM2909 ADR17,JSRFN0&AM2901 RAMD,ZB,,ØR,,R0
10          AM2909&AM2901          ,DA,,AND,R1,R1&DIN 1
11          AM2909 ADR17,JSRFN0&AM2901 RAMD,ZB,,ØR,,R1
12          AM2909&AM2901          ,DA,,AND,R2,R2&DIN 1
13          AM2909 ADR17,JSRFN0&AM2901 RAMD,ZB,,ØR,,R2
14          AM2909&AM2901 RAMF,ZB,CN0,SUBR,,R4
15          AM2909 ADR8,BRFN0&AM2901
16          AM2909 ADR18, BR&AM2901
```

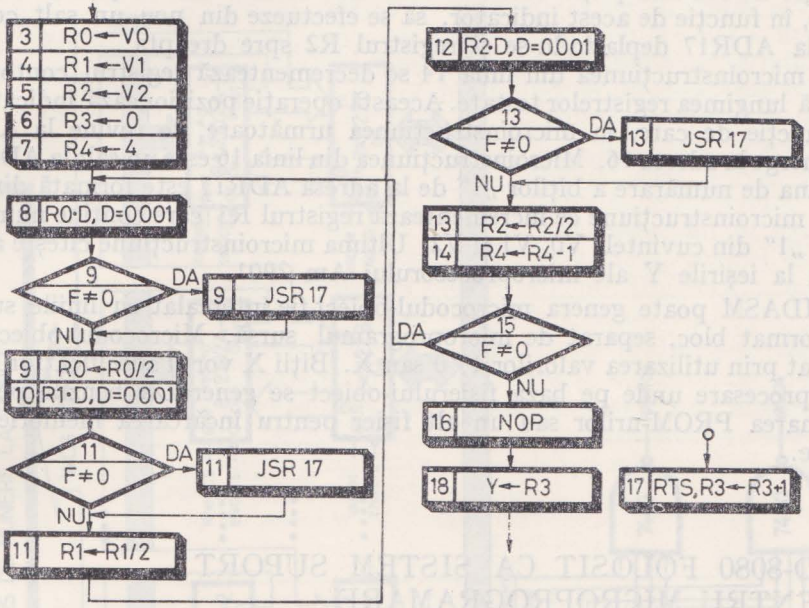


Fig. 8.4. O secvență de microprogram pentru unitatea centrală din figura 8.2.

```
17 ADR17: AM2909 ,RTS&AM2901 RAMF,ZB,CN1,ADD,,R3
18 ADR18: AM2909&AM2901      ,ZB,,ØR,,R3
19        END
```

După cum se vede, o linie de microinstrucțiune scrisă în AMDASM 29 este formată dintr-o etichetă opțională și mai multe definiții de format. Se observă substituirile opționale pentru cîmpurile variabile și operatorul „&" care reprezintă de fapt un SAU LOGIC folosit ca separator între definițiile de format.

În primele trei microinstrucțiuni, liniile 3, 4, 5, se încarcă V0 = 1111, V1=1001 și V2=0000 în registrele R0, R1 și respectiv R2. A patra microinstrucțiune șterge registrul contor R3, iar a cincea încarcă în registrul R4 mărimea cuvintelor V, aici 4. În microinstrucțiunea de la adresa ADR8, în vederea testării celei mai puțin semnificative cifre, se efectuează un ȘI LOGIC între conținutul registrului R0 și intrările directe D=0001. Această operație poziționează indicatorul de condiție Z din registrul de stare. Microinstrucțiunea următoare efectuează un salt condiționat la subrutina de la adresa ADR17 în funcție de valoarea indicatorului Z și deplasează dreapta cu o poziție registrul R0. Microinstrucțiunea de pe linia 10 poziționează, la fel ca mai sus, indicatorul Z în funcție de cea mai puțin semnificativă cifră binară a registrului R1. În continuare, în funcție de valoarea lui Z, se efectuează un salt condiționat la subrutina de la ADR17 și se deplasează dreapta registrul R1. Microinstrucțiunea de pe linia 12 poziționează indicatorul Z

în funcție de cea mai puțin semnificativă cifră binară din R2 pentru ca mai departe, în funcție de acest indicator, să se efectueze din nou un salt condiționat la ADR17 deplasîndu-se și registrul R2 spre dreapta.

În microinstrucțiunea din linia 14 se decrementează registrul contor R4 ce indică lungimea registrelor testate. Această operație poziționează indicatorul Z în funcție de care, în microinstrucțiunea următoare, se revine la ADR8 sau se merge la adresa 16. Microinstrucțiunea din linia 16 este un salt la ADR18. Subrutina de numărare a biților „1" de la adresa ADR17 este formată dintr-o singură microinstrucțiune ce incrementează registrul R3 care conține numărul de cifre „1" din cuvintele V0, V1 și V2. Ultima microinstrucțiune citește acest registru la ieșirile Y ale microprocesorului Am 2901.

AMDASM poate genera microcodul-obiect fie intercalat cu liniile sursă, fie în format bloc, separat de microprogramul sursă. Microcodul-obiect va fi generat prin utilizarea valorilor 1, 0 sau X. Biții X vor fi specificați în faza de postprocesare unde pe baza fișierului-obiect se generează benzile pentru programarea PROM-urilor sau un alt fișier pentru încărcarea memoriei de simulare.

8.2. SD-8080 FOLOSIT CA SISTEM SUPORT PENTRU MICROPROGRAMARE

În cap. 4 s-a menționat posibilitatea utilizării sistemului SD-8080 și ca sistem suport pentru microprogramare. În acest scop, autorii au construit un simulator pentru memoria de microprograme și un microasamblor dedicat unei anumite mașini microprogramate [5].

8.2.1. MEMORIA WCS PENTRU SIMULAREA MEMORIEI DE MICROPROGRAME

Aceasta este o memorie RAM de viteză medie care poate fi accesată pe octet de SD-8080 și pe cuvînt — microinstrucțiune — de mașina microprogramată care se pune la punct.

Structura simulatorului este dată în figura 8.5. După cum se vede, memoria WCS este construită cu circuite 2102A-2, 8226 și cîteva circuite integrate SSI și MSI cu ajutorul cărora s-a realizat logica de comandă. Circuitele de memorie 2102A-2, cele mai rapide din familia 2102, de 1k × 1 biți, au totuși un timp de acces destul de mare, de maximum 250 ns, ceea ce limitează utilizarea simulatorului la aplicații de viteză mică sau medie. Circuitele 8226 și 2102 au fost descrise în § 4.2.2.2, respectiv în § 4.2.3.2.

Mărimea memoriei WCS este de 16 Koctet pentru SD-8080 și de 1 K microinstrucțiuni de maximum 128 biți pentru mașina microprogramată care se pune la punct. Spațiul de memorie SD-8080 afectat acestei memorii este cuprins între 4000_{16} și $7FFF_{16}$. Adresele de microinstrucțiune sînt cuprinse între 000_{16} și $3FF_{16}$.

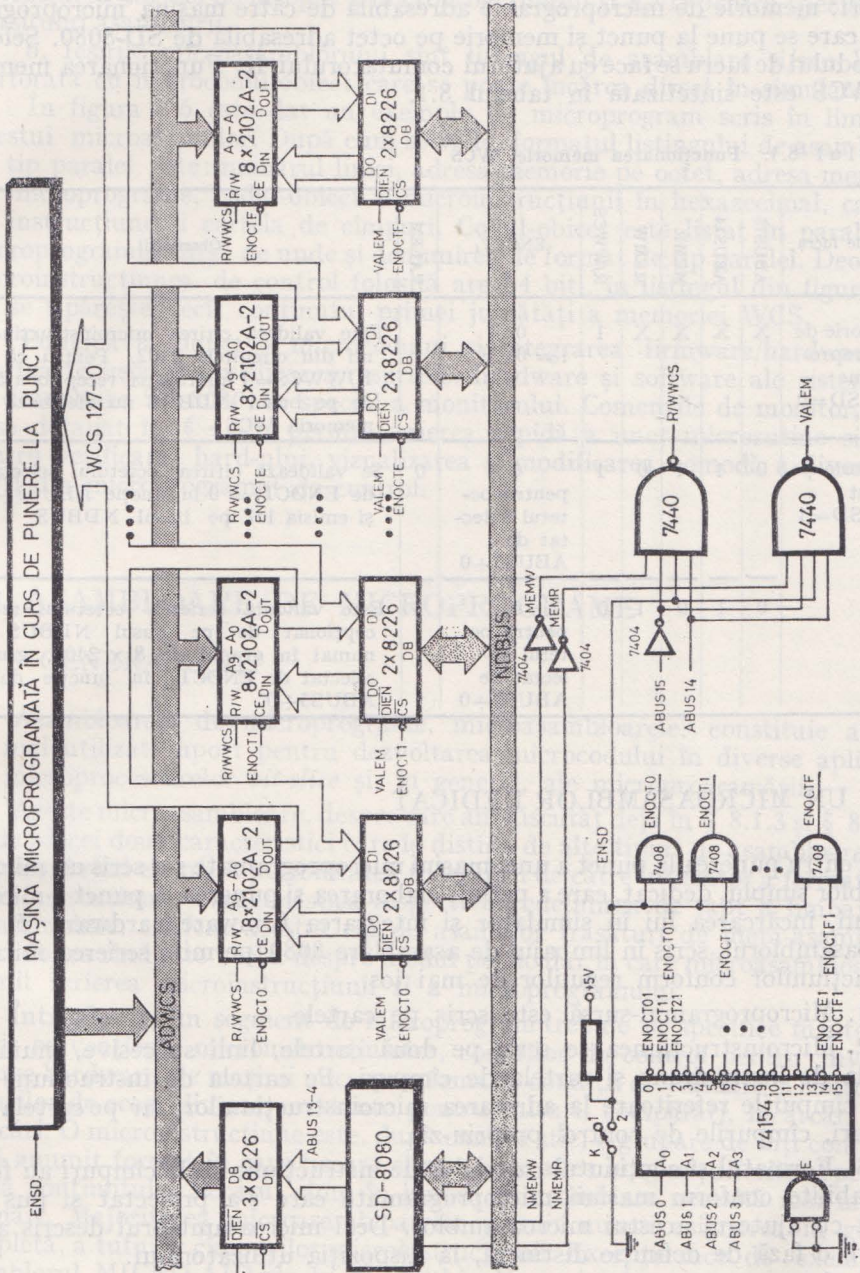


Fig. 8.5. Memoria WCS conectată la SD-8080

După cum se vede în figura 8.5 memoria WCS poate funcționa în două moduri: memorie de microprograme adresabilă de către mașina microprogramată care se pune la punct și memorie pe octet adresabilă de SD-8080. Selecția modului de lucru se face cu ajutorul comutatorului K. Funcționarea memoriei WCS este sintetizată în tabelul 8.1.

Tabelul 8.1. Funcționarea memoriei WCS

Mod de lucru	ABUS15	ABUS14	MEMR	MEMW	R/WWCS	ENØCT$_i$	VALEM	Observații
Memorie de microprograme ENSD=0	X	X	X	X	1	0 i = 0...F	1	Este validată citirea microinstrucțiunii din circuitele 2102. Pentru că R/WWCS=1 informația recepționată de pe busul NDBUS nu afectează memoria WCS
Memorie pe octet ENSD=1	0	1	1	0	1	0 pentru octetul selectat de ABUS3÷0	0	Se validează citirea octetului selectat de ENØCT$_i$ = 0 în funcție ABUS3÷0 și emisia lui pe busul NDBUS
	0	1	0	1	0	0 pentru octetul selectat de ABUS3÷0	1	Este validată scrierea octetului recepționat de pe busul NDBUS numai în grupul de 8 × 2102, grup selectat de ENØCT$_i$ în funcție de ABUS3÷0

8.2.2. UN MICROASAMBLOR DEDICAT

Pentru punerea la punct a unei mașini microprogramate s-a scris un microasamblor simplu, dedicat, care a permis elaborarea și punerea la punct a microcodului, încărcarea lui în simulator și integrarea firmware/hardware [5, 6] Microasamblorul, scris în limbajul de asamblare 8080, permite scrierea microinstrucțiunilor conform regulilor de mai jos.

1. Microprogramul-sursă este scris pe cartele.

2. Microinstrucțiunea se scrie pe două cartele, linii succesive, numite cartela de instrucțiune și cartela de cîmpuri. Pe cartela de instrucțiune se scriu cîmpurile referitoare la adresarea microinstrucțiunilor, iar pe cartela de cîmpuri, cîmpurile de control propriu-zis.

3. Formatul și conținutul cartelelor de instrucțiune și de cîmpuri au fost prestabilite conform mașinii microprogramate care s-a proiectat și pus la punct cu ajutorul acestui microasamblor. Deci microasamblorul descris aici nu are o fază de definiție distinctă, la dispoziția utilizatorului.

4. Între microinstrucțiuni pot fi intercalate cartele de comentariu. Aceste cartele încep cu „;" în prima coloană.

5. Microasamblorul are numai două directive de asamblare: ØRG pentru specificarea adresei de început a microprogramului și END pentru specificarea sfîrșitului asamblării.

6. Ieșirile microasamblorului sînt listingul de asamblare și/sau banda perforată cu microcodul-obiect care se poate încărca direct în simulator.

În figura 8.6 este dat un exemplu de microprogram scris în limbajul acestui microasamblor. După cum se vede, formatul listingului de asamblare, de tip paralel, este: numărul liniei, adresa-memorie pe octet, adresa-memorie de microprograme, codul-obiect al microinstrucțiunii în hexazecimal, cartela de instrucțiune și cartela de cîmpuri. Codul-obiect este listat în paralel cu microprogramul-sursă de unde și denumirea de format de tip paralel. Deoarece microinstrucțiunea de control folosită are 64 biți, în listingul din figura 8.6 nu se tipărește decît conținutul primei jumătăți a memoriei WCS.

Punerea la punct a microcodului și integrarea firmware/hardware se fac foarte ușor prin utilizarea resurselor hardware și software ale sistemului de dezvoltare SD-8080, în special a monitorului. Comenzile de monitor, descrise detaliat în § 4.3.1, permit scrierea rapidă a unor microrutine simple pentru verificarea hard-ului, vizualizarea și modificarea comodă a diverselor părți din microprogramul de control.

8.3. ASAMBLOARE DE MICROPROGRAME

8.3.1. GENERALITĂȚI

Asambloarele de microprograme, microasambloarele, constituie astăzi cel mai utilizat suport pentru dezvoltarea microcodului în diverse aplicații ale microprocesoarelor *bit-slice* și, în general, ale microprogramării.

Aceste microasambloare, despre care am discutat deja în § 8.1.3 și § 8.2.2, au de obicei două caracteristici care le disting de alte tipuri de asambloare [8]. În primul rînd ele oferă posibilitatea redefinirii de către utilizator a formatului microinstrucțiunii. Caracteristica respectivă, redefinirea de fapt a limbajului de asamblare, conferă acestui tip de asamblor trăsături de meta-asamblor. În al doilea rînd este vorba despre modul particular în care microasambloarele permit scrierea microinstrucțiunii și a microprogramului.

Într-adevăr, un segment de microprogram trebuie să specifice multe lucruri: secvențierea microinstrucțiunilor, codurilor de control pentru diverse resurse hardware ale mașinii microprogramate, condiții de validare și controlul ciclurilor de ceas, diverse constante pentru salturi, comparări, încărcări sau mascări. O microinstrucțiune este, după cum se știe, o grupare de biți conform unui anumit format în care fiecare cîmp poate avea lungimi diferite. Specificarea conținutului fiecărui cîmp în limbaj de asamblare necesită mai multe asignări. Majoritatea microasambloarelor permit numai asignarea în grup, completă, a tuturor cîmpurilor microinstrucțiunii. Excepție face, de exemplu, asamblorul MICRO-AID de la Monolithic Memories în care programatorul specifică o microinstrucțiune precizînd pe linii-sursă separate cîmpurile utili-

```
0016                                    ;
0017                                    ;
0018                                    ;              ASTEPTARE INSTRUCTIUNI UC
0019                                    ;
0020                                    ;
0021                                    ;SE ASTEAPTA O INSTRUCTIUNE DE LA UNITATEA CENTRALA.DUPA RECEPTIONAREA TUCS
0022                                    ;SE DECODIFICA CI0,1,2 SI SE SARE LA SUBRUTINA CORESPUNZATOARE.IN CAZUL
0023                                    ;UNUI COD DE INSTRUCTIUNE INCORECT SE BLOCHEAZA UC(ATUC=0).
0024                                    ;ZS DIN MICROPROGRAM SINT NEGATE
0025                                    ;
0026                                    ;
0027 4000 0000 00 00 03 60 00 06 00 00 ZERO     JSRK    ZERO            TUCS
0028
0029 4010 0001 00 00 03 A0 00 1A 00 00          JMPK    ZERO    TUCS YCI0
0030
0031 4020 0002 00 00 13 A0 00 42 40 00          JMPK    ZERO1   YCI1 YCI2
0032
0033 4030 0003 00 00 23 A0 00 42 40 00          JMPK    ZERO2   YCI1 YCI2
0034
0035 4040 0004 00 00 03 A0 00 00 00 00 ZERO1    JMPK    ZERO
0036
0037 4050 0005 00 00 03 A0 00 00 00 00          JMPK    ZERO
0038
0039 4060 0006 00 00 33 B0 0D 00 08 28          JMPK    SIO
0040                                                    SAD40           FCA7            YDAM3YDAM4
0041 4070 0007 00 01 43 B0 0D 00 08 28          JMPK    HIO
0042                                                    SAD40.          FCA7            YDAM3YDAM4
0043 4080 0008 00 00 B3 B0 0D 00 08 28 ZERO2    JMPK    TIO
0044                                                    SAD40           FCA7            YDAM3YDAM4
0045 4090 0009 00 01 C3 B0 0D 00 08 28          JMPK    TDV
0046                                                    SAD40           FCA7            YDAM3YDAM4
0047 40A0 000A 00 02 33 A0 00 00 C0 00          JMPK    AIO             YDI
0048
0049 40B0 000B 00 00 03 A0 00 00 00 00          JMPK    ZERO
```

Fig. 8.6. Un listing generat de microasamblorul dedicat rulat pe SD-8080

zate. În final aceste linii se vor asambla într-o singură microinstrucțiune. Astfel, o linie tipică de microprogram diferă de o instrucțiune-mașină convențională, de exemplu de tipul 8080, prin aceea că ea conține mai multe coduri-operație.

Asambloarele de microprograme trebuie deci să îndeplinească cerințe deosebite: redefinirea ușoară a mașinii *target* — a microinstrucțiunii, scrierea comodă a microinstrucțiunilor, încărcarea, în general comoditate în utilizare. În continuare vom încerca să detaliem aceste cerințe.

Deoarece microasambloarele sînt mijloace suport pentru proiectarea și punerea la punct hardware/firmware, ele trebuie să permită schimbări ale mașinii *target* aproape la fiecare utilizare a lor. Cea mai obișnuită metodă de a implementa această cerință este aceea prin care programul este împărțit în două faze distincte: o fază de definire și una da asamblare propriu-zisă.

În faza de definire se specifică formatul microinstrucțiunii precizîndu-se mărimea, poziția și valorile în lipsă pentru fiecare cîmp. De asemenea, în această fază se definesc toate codurile mnemonice asociate microoperațiilor ce vor fi utilizate în faza următoare. Definițiile trebuie să se scrie simplu și să poată fi introduse ca fișiere pe orice terminal, cititor de cartele sau disc. În faza de definire asamblorul va face o analiză gramaticală a tuturor definițiilor și va construi pe baza lor o descriere internă a mașinii *target*.

Faza de asamblare este asemănătoare cu procesarea pe care o face un asamblor de calculator cu arhitectură fixă. În această fază un microasamblor explorează liniile-sursă translatînd și asamblînd codurile mnemonice, expresiile, alte simboluri, în microinstrucțiuni binare. Pentru a fi ușor de utilizat un microasamblor va trebui să respecte, pe cît posibil, regulile generale ale limbajelor de asamblare. Astfel, o linie tipică de microasamblor poate începe cu o etichetă alfanumerică urmată de o serie de coduri mnemonice. Ca separatori se recomandă, de exemplu, blancuri sau virgule. Deoarece microinstrucțiunile conțin uneori multe microoperații ele trebuie să poată fi scrise pe mai multe linii. Atunci cînd este cazul, cîmpurile care au specificată în faza de definire valoarea în lipsă pot fi omise în liniile-sursă. Comentariile sînt permise, dar uneori ele sînt restrînse în porțiunea rămasă la sfîrșitul liniei, după microinstrucțiune.

După fazele de definire a arhitecturii și de scriere a microprogramului urmează o așa-numită fază de postprocesare, de încărcare, în care se pregătește ieșirea microasamblorului pentru ca microcodul-obiect să poată fi încărcat în memoria de microprograme. Ieșirea trebuie să corespundă accstei memorii care poate fi de tip WCS, PROM sau PLA.

Utilizarea comodă a microasambloarelor este condiționată de mai mulți factori: microasambloarele trebuie să fie cît mai simple ca sintaxă și operare, să permită specificarea ușoară a valorilor în lipsă sau a altor opțiuni. Este de asemenea important ca utilizarea unui microasamblor să se poată face pe mai multe niveluri de complexitate pornind de la microprograme pentru mașini foarte simple și ajungînd la mijloace avansate de programare, cum sînt de exemplu macrodefinițiile. Un alt factor care influențează utilizabilitatea unui microasamblor este și posibilitatea ca acesta să poată fi folosit local, ceea ce va permite ca microcodul să fie ușor încărcat în mașina microprogramată care se pune la punct. Unele microasambloare pot fi utilizate în time-sharing,

altele pot fi obținute în limbaj-sursă, în general FORTRAN, pentru a fi rulate pe calculatoare proprii. Totuși, se pare că cea mai bună soluție este de a avea un microasamblor care să poată fi rulat pe un sistem de dezvoltare. Astăzi, cînd microprocesoarele au ajuns într-adevăr cuvînt de ordine pentru proiectanții de structuri de control, majoritatea laboratoarelor de proiectare electronice ar fi util să aibă cîte un sistem de dezvoltare. Acest sistem, destinat în principal dezvoltării structurilor cu microprocesoare *single-chip*, poate fi, în majoritatea cazurilor, adaptat și pentru proiectarea structurilor cu microprocesoare *bit-slice*.

În tabelul 8.2 sînt descrise cîteva din cele mai cunoscute asambloare de microprograme.

Tabelul 8.2. **Asambloare de microprograme (microasambloare)**

Numele microasamblorului (firma producătoare)	Caracteristici generale
AMDASM (Advanced Micro Devices)	Meta-asamblor în două faze separate: definire și asamblare propriu-zisă. Separarea în două faze este deosebit de utilă deoarece definirea microinstrucțiunii se face, în general, de mai multe ori, înainte de scrierea microprogramelor. Se poate utiliza în time-sharing sau pe sistemele de dezvoltare *System* 29 și MDS. Este scris în FORTRAN V și PL/M-80.
CROMIS (Intel)	Cel mai puțin flexibil asamblor, permite proiectarea structurilor cu microprocesoare *bit-slice* din familia 3 000. CROMIS constă din limbajul de microasamblare XMAS și limbajul de *mapare* a memoriilor PROM, XMAP. Este scris în FORTRAN IV.
DAPL (Zeno Systems)	Limbajul, scris în PL/1 pentru calculatoare Burroughs, este organizat pe patru niveluri de complexitate, fiind atractiv și pentru că are un set clar de instrucțiuni ce pot fi folosite pînă la nivelul de macrodefiniții. Ca dezavantaj se menționează că DAPL este utilizabil numai în time-sharing.
MICRO-AID (Monolithic Memories)	Este scris în FORTRAN IV și are o sintaxă în general simplă, foarte asemănătoare cu a asambloarelor convenționale. Definițiile pot fi memorate într-un fișier sau pot fi incluse în secțiunea de asamblare.
RAPID (Scientific Micro Systems)	Cel mai ușor obtenabil microasamblor. Fiind scris în FORTRAN IV, este disponibil atît în time-sharing, cît și separat, fiind adaptat și pentru minicalculatoare de tip Nova. Totuși limbajul este oarecum ciudat și într-un fel incomod.
RAYASM (Raytheon)	Asamblorul este scris în PL/1. Are o varietate deosebită de instrucțiuni de definire. Prezintă dezavantajele că limbajul este uneori nenatural și microinstrucțiunile sînt asamblate întotdeauna pe 95 biți.
Signetics Micro-Assembler (Signetics)	Scris în FORTRAN IV, acest meta-asamblor permite, ca și AMDASM, scrierea microprogramelor în două faze separate: definițiile și microprogramul propriu-zis. Conține instrucțiuni și directive opționale care îl apropie de limbajele de nivel înalt.

Din cele spuse mai sus putem concluziona că microasambloarele sînt cu atît mai utile cu cît răspund mai bine următoarelor cerințe:

— definirea microinstrucțiunii, separată sau neseparată de faza de asamblare, trebuie să permită utilizatorului suficientă libertate în specificarea cîmpurilor și codurilor mnemonice, fără o complicare exagerată a limbajului;

— faza de asamblare, pentru a fi ușor de folosit, trebuie să fie cît mai asemănătoare cu limbajele de asamblare moderne, pe cît posibil cu limbajele de nivel înalt;

— este deosebit de util ca un microasamblor să aibă o fază de postprocesare, pentru a asigura reorganizarea microcodului-obiect, astfel încît acesta să poată fi direct încărcat în simulatoare sau ars în memoriile PROM.

Pe parcursul dezvoltării mijloacelor de programare limbajele de microprogramare au urmărit în general limbajele de programare. Deși în ultimii ani s-au făcut progrese importante în ceea ce privește limbajele de microprogramare, în acest domeniu, al proiectării cu microprocesoare *bit-slice*, majoritatea mijloacelor suport de tip software sînt totuși de nivel scăzut sau chiar primitiv. S-au elaborat numai sporadic limbaje mai evoluate care pot utiliza tehnica macro-definițiilor sau notații pentru transferul între registre și, mai rar, limbaje de nivel înalt.

În [8] se estimează că dezvoltarea firmware este, în general, mai costisitoare decît cea software și că soluția ar fi elaborarea unor limbaje de nivel înalt pentru microprogramare, a unor sisteme sau limbaje pentru descrierea hardware-ului. Problema este deosebit de complicată și ea ține de fapt de elaborarea unor sisteme și limbaje evoluate destinate proiectării diverselor structuri de control numerice.

8.3.2. META-ASAMBLORUL METASM

8.3.2.1. Caracteristici generale

Se știe că asambloarele, poate mai corect programele de asamblare simbolică, sînt sisteme care asistă programatorul în pregătirea programelor în cod-mașină [10]. În general, majoritatea acestor asambloare trebuie să organizeze tabele de simboluri, să evalueze expresii și să genereze cuvinte binare corespunzătoare unor cîmpuri simbolice. Un meta-asamblor reprezintă un sistem care poate asigura toate aceste cerințe, avînd în plus mijloace independente de mașină pentru descrierea unor diverse reguli de asamblare. Un meta-asamblor acceptă această descriere, stabilirea limbajului, după care funcționează ca un asamblor obișnuit.

Se pare că ideea unui meta-asamblor a avut-o pentru prima oară D.E. Ferguson, care în 1966 a publicat un articol despre acest tip de asamblor [10]. Ideile enunțate în acest articol au fost utilizate în asambloarele METASYMBOL, pentru seria de calculatoare SDS 900, și în asambloarele SLEUTH II și UTMOST, pentru calculatoarele UNIVAC 1107/8.

În § 8.3.1 am menționat că microasambloarele au în general posibilități de definire a mașinii *target*, ceea ce le conferă aspectul de meta-asambloare. Meta-asamblorul METASM, pe care îl descriem aici, a fost conceput și realizat

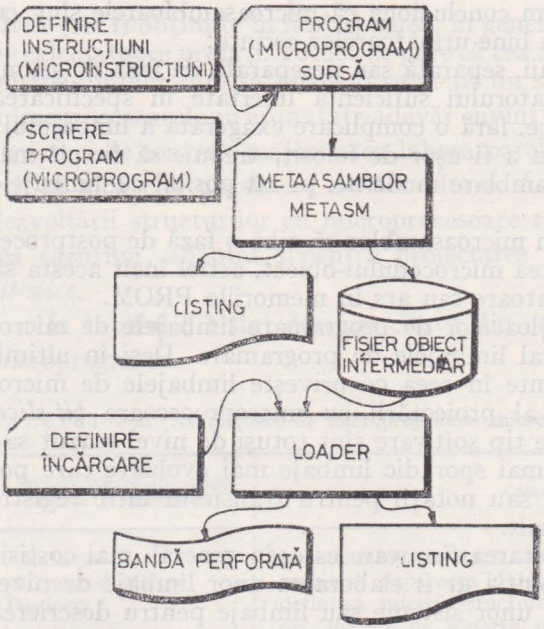


Fig. 8.7. Schema de utilizare a meta-asamblorului METASM

pentru a fi un mijloc suport cu posibilități extinse pentru dezvoltarea unei game cît mai largi de structuri de control cu microprocesoare.

METASM, a cărei schemă de utilizare este dată în figura 8.7, este un program scris în FORTRAN. Meta-asamblorul asigură scrierea de programe pentru o mare diversitate de structuri de control, construite atît cu microprocesoare de tip *bit-slice*, cît și cu cele de tip *single-chip*. În acest fel METASM poate fi folosit în aplicații ale microprogramării, pentru elaborarea microprogramelor de control, precum și pentru scrierea de programe în limbaje de asamblare, de exemplu de tipul 8080 [6, 9].

METASM permite în prima secțiune a programului-sursă utilizator specificarea microinstrucțiunii sau a instrucțiunii, apoi în a doua secțiune, scrierea microprogramelor, respectiv a programelor, în limbajul definit anterior.

Cartelele de comandă și directivele prin care programatorul comunică cu METASM sînt asemănătoare cu cele ale Micro Assembler-ului elaborat de Signetics [11].

În continuare vom descrie cîteva din elementele de bază ale limbajului METASM [12]. Formatul utilizat pentru descrierea cartelelor de comandă și a directivelor meta-asamblorului este următorul:

$$[\text{eticheta:}]\ \begin{matrix}\text{CØD}\\ \text{ØPERAȚIE}\end{matrix}\ \{\text{nume s}\}\left\{\begin{matrix}\text{CUVÎNT}\\ \text{CHEIE}\end{matrix}\ \begin{matrix}\text{s}\\ \text{c}\\ \text{e}\end{matrix}\right\}[\text{nume e}]\ \ldots;\ [\text{CØMENTARIU}] \tag{8.1}$$

unde:

[ ]	parantezele drepte indică un simbol sau o altă parte opțională a cartelei de comandă sau a directivei;
CØD ØPERAȚIE	reprezintă numele propriu-zis al cartelei de comandă sau al directivei și este scris întotdeauna cu litere mari;
{ }	acoladele mici indică un operand asociat CØD-ului-ØPERAȚIE al cărui nume va fi scris cu litere mici;
{ CUVÎNT CHEIE s c e }	un cuvînt cheie este un cuvînt rezervat, cuprins întotdeauna între acolade și scris cu litere mari, care comunică asamblorului că urmează un parametru specific;
nume	este un nume de operand scris cu litere mici și cuprins între acolade sau paranteze drepte;
nume e	numele operandului este urmat de o literă semnificînd ce element de limbaj poate fi utilizat;
s	simbol;
c	constantă;
e	expresie;
s	sublinierea precizează faptul că elementul a fost definit anterior în programul-sursă;
s c e	indică posibilitatea unei selecții între s, c și e;
...	operandul respectiv poate fi repetat;
;	marchează sfîrșitul unei cartele de comandă, al unei directive sau al unei microinstrucțiuni și/sau începutul comentariului.

8.3.2.2. Definirea limbajului de asamblare

În acest paragraf ne vom referi la modul de definire al microinstrucțiunilor. Pe parcursul proiectării unei mașini microprogramate METASM poate fi folosit după stabilirea formatului microinstrucțiunii. În secțiunea de definire proiectantul va trebui să construiască un limbaj de asamblare specificînd toate informațiile necesare despre microinstrucțiunea pe care o utilizează. Aceste informații cuprind mărimea microinstrucțiunii, numele mnemonice ale tuturor cîmpurilor, mărimea cîmpurilor și poziția lor în cadrul microinstrucțiunii, numele mnemonice și valorile tuturor microoperațiilor, valorile în lipsă pe care asamblorul le va folosi atunci cînd cîmpul respectiv nu este menționat în linia de microinstrucțiune.

Definirea unei microinstrucțiuni se face cu ajutorul unei secvențe de cartele de comandă. Această secvență este alcătuită dintr-o cartelă de comandă DEFIN, mai multe cartele de comandă FIELD și cartela de comandă ENDDEF, în această ordine.

Cartela de comandă DEFIN specifică mărimea microinstrucțiunii. Formatul ei, scris conform cu (8.1), este următorul:

$$\text{DEFIN}\left\{\text{WIDTH}\begin{matrix}\text{s}\\ \text{c}\\ \text{e}\end{matrix}\right\};\ [\text{CØMENTARIU}]$$

Exemplu de utilizare:

```
DEFIN WIDTH 64; LUNGIMEA MICRØINSTRUCȚIUNII
```

Cartela de comandă FIELD asignează un nume mnemonic unui cîmp și precizează numărul de biți ai cîmpului. Opțional se poate specifica valoarea în lipsă a cîmpului. Formatul cartelei de comandă FIELD este următorul:

$$\text{FIELD }\{\text{nume cîmp s}\}\left\{\text{WIDTH}\begin{matrix}\text{s}\\ \text{c}\\ \text{e}\end{matrix}\right\}\left[\text{DEFAULT}\begin{matrix}\text{s}\\ \text{c}\\ \text{e}\end{matrix}\right];\ [\text{CØMENTARIU}]$$

Exemplu de utilizare:

```
FIELD MCAU WIDTH 5 DEFAULT 05H; CØNTRØLUL ADRESEI
;URMĂTØARE
```

Cîmpurile sînt plasate în microinstrucțiune conform ordinii în care au fost definite în programul-sursă. De exemplu, cartelele de comandă:

```
DEFIN WIDTH 16;
FIELD ØP WIDTH 6;
FIELD R WIDTH 2;
FIELD DATA WIDTH 8;
```

definesc microinstrucțiunea din figura 8.8.

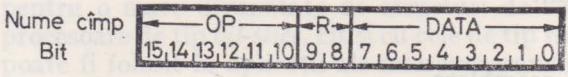


Fig. 8.8. O microinstrucțiune definită cu METASM

Definirea microinstrucțiunii se încheie cu cartela de comandă ENDDEF care are formatul:

```
ENDDEF; [CØMENTARIU]
```

După definirea microinstrucțiunii programatorul poate începe scrierea microprogramului. O microinstrucțiune se va reprezenta simbolic printr-o succesiune de nume de cîmpuri, fiecărui nume fiindu-i asociat un operand care îi va specifica valoarea pentru microinstrucțiunea respectivă.

De exemplu, definind următoarele coduri mnemonice:

```
NØP:  EQU  00Q;  „NØP"
ADD:  EQU  01Q;  ADUNARE
R0:   EQU  0;    REGISTRUL R0
```

putem scrie microinstrucțiunea:

```
EX1: ØP=ADD R=R0 DATA=07H; R0=R0+07H
```

Meta-asamblorul METASM permite însă scrierea microinstrucțiunilor utilizînd un nivel superior de definire și anume prin intermediul cartelei de comandă MICRØP. *Microp*-ul este un nume mnemonic asociat valorilor unuia sau mai multor cîmpuri din microinstrucțiune. Deci prin intermediul cartelei de comandă MICRØP se poate „personaliza" o parte sau chiar toată microinstrucțiunea. *Microp*-urile ușurează atît scrierea, cît și înțelegerea microprogramelor reprezentînd o facilitate deosebit de utilă în proiectarea microcodului.

Formatul cartelei de comandă MICRØP este următorul:

$$\text{MICRØP}\ \{\text{nume } \textit{microp}\ \text{s}\}\ \text{ASSIGN}\ \left\{\text{nume cîmp} = \begin{matrix}\text{s}\\ \text{c}\\ \text{e}\end{matrix}\right\}\ \ldots\ ;\ [\text{CØMENTARIU}]$$

Exemple de utilizare:

```
MICRØP ADD ASSIGN ØP=ADD;
MICRØP CØNTINUE ASSIGN MCAU=CØNT;
MICRØP WAITR ASSIGN MSCB=SNDAMS MØCT=ØCT00
MCDA=FCAZ;
```

Atunci cînd METASM întîlnește în microprogramul-sursă un *microp* el va substitui întreaga listă de valori cîmpurilor personalizate de *microp*-ul respectiv.

Deci, în concluzie, după cartela ENDDEF, care marchează sfîrșitul definirii microinstrucțiunii, urmează, de obicei, definirea codurilor mnemonice ce se poate face cu ajutorul directivelor EQU, SET și al cartelei de comandă MICRØP. După aceste definiții se va scrie microprogramul.

8.3.2.3. Scrierea microprogramului

Secțiunea de microprogram specifică, la început, cu ajutorul cartelei de comandă MPRØG, dimensiunile memoriei de control în care va fi încărcat microcodul; urmează microprogramul propriu-zis scris ca o succesiune de microinstrucțiuni.

Microinstrucțiunea trebuie să precizeze valoarea fiecărui cîmp de control prin utilizarea definițiilor din secțiunea de definire. METASM va asambla fiecare microinstrucțiune producînd o linie de text-obiect intermediar. Ieșirile METASM sînt următoarele:

— tabele cu toate simbolurile utilizate în microprogramul-sursă;
— un listing care conține microprogramul-sursă și codul-obiect;
— microprogramul asamblat ca text-obiect intermediar.

Textul-obiect intermediar va fi utilizat mai departe ca sursă-pentru un program de tip *loader*. Acest *loader* prelucrează textul-obiect intermediar conform unor cartele de comandă specifice și generează în final codul-obiect definitiv al microprogramului asamblat. Codul-obiect final poate fi încărcat direct într-o memorie WCS sau ars în PROM-uri.

În timpul asamblării METASM administrează numărătorul de locații de microprogram. Acest numărător de locații este inițializat pe zero de cartela MPRØG după care va fi incrementat de fiecare microinstrucțiune. Valoarea curentă a numărătorului de locații poate fi alterată prin directiva ØRG.

Prima cartelă de comandă din secțiunea de microprogram, MPRØG, are formatul:

$$\text{MPRØG } \{\text{nume s}\} \left\{ \text{WIDTH} \begin{matrix} \underline{s} \\ c \\ e \end{matrix} \right\} \left[\text{LENGTH} \begin{matrix} \underline{s} \\ c \\ e \end{matrix} \right] ; \text{ [CØMENTARIU]}$$

Exemplu de utilizare:

```
MPRØG MPCS WIDTH 64; MICRØPROGRAM DE CØNTRØL PENTRU
;CS
```

Scrierea microinstrucțiunilor se face cu formatul următor:

$$\text{[eticheta]: } \left\{ \begin{matrix} \text{nume cîmp } \underline{s} = \begin{matrix} s \\ c \\ e \end{matrix} \\ \text{nume } \textit{microp } \underline{s} \end{matrix} \right\} \ldots ; \text{ [CØMENTARIU]}$$

După cum se vede microinstrucțiunea nu are un CØD ØPERAȚIE specific. METASM recunoaște o microinstrucțiune, atunci cînd întîlnește un nume de cîmp sau de *microp*. După eticheta opțională urmează o listă de nume de cîmpuri și/sau de *microp*-uri împreună cu operanzii asociați. O microinstrucțiune trebuie să specifice valorile tuturor cîmpurilor utilizînd operanzi de cîmp, *microp*-uri sau valori în lipsă.

Exemple de microinstrucțiuni:

```
ZERØ:MCAU=JMPK MK=ZERØ MSCB=STUCSB; TESTARE TUC
TIØ: MCAU=JMPK MK=TIØ WAITR MSADR=SAD40; AȘTEPTARE
;CITIRE
```

Secțiunea de microprogram se încheie cu cartela de comandă END care are formatul:

```
END; [CØMENTARIU]
```

Directivele meta-asamblorului METASM sînt cele obișnuite de asamblare (EQU, SET, ØRG, DS, DB, DW) și de tipărire (LIST, SPACE, EJECT, TITLE). Aceste directive sînt prevăzute pentru a ușura elaborarea microprogramelor și documentarea lor. Ele pot fi intercalate liber printre microinstrucțiuni, oriunde în secțiunea de microprogram.

8.3.2.4. Facilități extinse ale meta-asamblorului METASM

Puterea de utilizare a meta-asamblorului METASM poate crește prin folosirea facilităților extinse. Ne referim aici la extinderea repertoriului de operatori, la posibilitatea de programare a memoriilor PROM care nu conțin

microprograme, la *microp*-urile cu argumente și *microp*-urile înlănțuite, la clauza IF THEN ELSE utilizabilă în definirea *microp*-urilor, la definițiile de subcîmpuri, la definițiile multiple de cîmp, la definițiile multiple de format.

Vom da mai jos două exemple de utilizare a acestor facilități extinse.

Instrucțiunile microprocesoarelor *single-chip* pot fi specificate cu ajutorul definirilor multiple de format și *microp*-ului cu argumente. Am spus că METASM permite definiri multiple de format. Acest lucru înseamnă că în secțiunea de definire se pot preciza mai multe tipuri de microinstrucțiuni. Definirea fiecărui tip de microinstrucțiune necesită o cartelă de comandă DEFIN, mai multe cartele FIELD, cartela ENDDEF, directive și *microp*-uri.

Ca exemplu instrucțiunea MØV R1, R2 a microprocesorului 8080 se poate specifica, presupunînd formatul definit, astfel:

```
MICRØP MØV R1, R2 ASSIGN ØP=01B DEST=R1 SURSA=R2;
```

Utilizarea acestui microp se face foarte simplu:

```
MØV A,B; A ← B
```

Analog se pot defini toate instrucțiunile 8080, ceea ce va permite utilizarea METASM și pentru asamblarea programelor scrise în limbajul de asamblare 8080.

O altă facilitate extinsă este aceea că *microp*-urile pot fi utilizate înlănțuit. De exemplu:

```
MICRØP MEMØN DIRECTIØN ASSIGN MEM=DIRECTIØN;
MICRØP DBUSØN DIRECTIØN ASSIGN DB=DIRECTIØN;
IN:EQU 01B;
ØUT:EQU 10B;
MICRØP FETCH ASSIGN MEMØN ØUT DBUSØN IN;
```

Pentru o înțelegere mai clară a modului de utilizare a METASM în elaborarea microprogramelor vezi și § 10.1.3.3. și § 10.2.

BIBLIOGRAFIE

1. DAVIDSON, S.; SHRIVER, B.D., *An Overview of Firmware Engineering*, Computer, 1978, 11, 5, p. 21–33.
2. ROYBAL, PH., *A microprogram development system*, National Semiconductor Corp., 1974.
3. BALPH, T.; BLOOD, W., *Assembler streamlines microprogramming*, Computer Design, 1979, 18, 12, p. 78–89.
4. * * * *The Am 2900 Family Data Book With Related Support Circuits*, Advanced Micro Devices, Inc., 1979 p. 3.4–3.25.
5. LUPU, C., *Microprogramarea modernă*, Primul simpozion național de teoria sistemelor, Craiova, 1980, vol. II, p. 273–279.
6. LUPU, C.; DIMITRIU, B.; IONESCU, C., *Sisteme suport pentru dezvoltarea structurilor de control microprogramate*, Al VII-lea Simpozion „Informatică și conducere", Cluj-Napoca, 1981.

7. * * * *Universal Development System Offers Real Time Emulation of 4-to 32-Bit Microprocessors*, Computer Design, 1980, 19, 9, p. 166—168.
8. POWERS, V.M; HERNANDEZ, J.H., *Microprogram Assemblers for Bit-Slice Microprocessors*, Computer, 1978, 11, 7, p. 108—120.
9. LUPU, C.; STĂNCESCU, ȘT.; DIMITRIU, B.; IONESCU C., *Metodologii de proiectare a structurilor de control numerice*, 4th International Conference on Control Systems and Computer Science, Bucharest, 1981, vol. II, p. 60—65.
10. BARRON, D.W., *Assemblers and Loaders*, MacDonald: London and American Elsevier Inc., New York, 1969.
11. * * * *Signetics Micro Assembler Reference Manual*, Signetics Corp., Sunnyvale, California, 1977.
12. * * * *Meta-asamblorul CCAB ASM — Manual de utilizare*, Centrul de cercetări de automatică, București, 1981.
13. * * * *STEP 2, a bit-slice development system*, Electronic Design, 1980, July 19.
14. SCRUPSKI, S.E., *Universal System emulates to 30 MHz*, Electronic Design, 1980, 28, 13, p. 31—32.
15. * * * *System emulates 35-ns memory*, Electronics, 1980, 53, 24, p. 214—215.
16. * * * *Unit develops wide bit-slice CPUs*, Electronics, 1980, 53, 20, p. 218.

MICROPROCESOARE *BIT-SLICE*

9.1. GENERALITĂȚI

În acest capitol vom descrie cîteva din cele mai importante circuite ale cunoscutei familii de microprocesoare *bit-slice* 2900. De asemenea vor fi descrise, pe scurt, familiile *Intel* 3000, F100K, M10800, MACROLOGIC, MMI5700/6700, TI74481/482.

Menționăm că familiile de circuite *bit-slice* sînt alcătuite din diverse blocuri constructive care au rolul de a permite implementarea cu ușurință a structurilor de control microprogramate. Așa cum s-a spus în cap. 7, astfel de structuri au două funcții principale — funcția de secvențiere și funcția de procesare propriu-zisă, de control — împărțite la rîndul lor în următoarele subfuncții [1]:

— procesarea datelor;
— controlul adresei de microprogram;
— controlul adresei de macroprogram;
— controlul întreruperilor;
— accesul direct la memorie (DMA);
— controlul I/E;
— controlul memoriei.

Circuitele fiecărei familii trebuie să asigure implementarea comodă a subfuncțiilor. Din acest punct de vedere familia cea mai completă, cea mai diversificată, este 2900. Apreciem că diversificarea acestei familii a condus la deosebita popularitate de care ea se bucură în rîndul proiectanților de structuri de control microprogramate. Familia cuprinde foarte multe circuite specializate destinate realizării unor subfuncții de tipul celor enumerate mai sus, necesare într-o SCM. De exemplu, procesarea datelor poate fi asigurată de microprocesoarele *bit-slice* 2901, 2903, 29203 sau 29116, împreună cu generatoarele de transport anticipat de tipul 2902 și circuitele 2904 pentru controlul operațiilor de deplasare și indicatorilor de condiție. Controlul adresei de microprogram este asigurat de secvențiatoarele de microprogram 2909/2911 sau 2910, iar controlul adresei de macroprogram, de controlorul de program 2930. Controlul întreruperilor poate fi implementat cu ajutorul circuitelor de tipul 2914, accesul direct la memorie — cu circuitul 2940, I/E — cu porturile 2950/2951, controlul memoriei cu controlorul de memorie dinamică 2964, iar cu circuitul 2960 detecția și corecția erorilor. Celelalte familii menționate mai sus asigură, în general, numai implementarea unor funcții de bază ale SCM, cum sînt funcția de secvențiere și/sau cea de procesare aritmetică.

Tabelul 9.1. **Principalele familii de microprocesoare** *bit-slice*

Familia (Fabricant principal)	Tehnologia	Frecvența de lucru maximă	Cuvîntul cu care lucrează	Compatibilitatea TTL	Software de dezvoltare	Observații
2900 (AMD)	STTL	10 MHz	4 Biți	Da	Da	Cea mai dezvoltată familie, cei mai mulți furnizori
MACROLOGIC (FAIRCHILD)	STTL/CMOS	10/2 MHz	4 Biți	Da	Da	Versiune CMOS pentru aplicații de putere mică
F100K (FAIRCHILD)	ECL	50 MHz	8 Biți	Nu	Da	*Bit-slice* pe 8 biți, cea mai rapidă familie
3000 (INTEL)	STTL	10 MHz	2 Biți	Da	Da	Prima familie de microprocesoare *bit-slice*, circuite pe 2 biți
M10800 (MOTOROLA)	ECL	20 MHz	4 Biți	Nu	Da	Cea mai cunoscută familie de microprocesoare ECL
SBP-0400 (TEXAS INSTRUMENTS)	I^2L	5 MHz	4 Biți	Da	Nu	Microprocesor *bit-slice* în tehnologie I^2L
74481/482 (TEXAS INSTRUMENTS)	STTL/LSTTL	10 MHz	4 Biți	Da	Da	Microprocesor *bit-slice* cu un set deosebit de bogat de instrucțiuni

În tabelul 9.1 se prezintă sintetic principalele familii de circuite *bit-slice* indicîndu-se tehnologia de fabricație, frecvența maximă de lucru, mărimea cuvîntului, „feliei", procesate, compatibilitatea TTL, asigurarea cu mijloace software pentru dezvoltare.

9.2. FAMILIA Am 2900

9.2.1. MICROPROCESORUL *BIT-SLICE* Am 2901

Acest microprocesor *bit-slice* de 4 biți este conceput ca bloc constructiv de mare viteză și este destinat proiectării de unități centrale, de dispozitive de control al I/E, în general, implementării de structuri de control microprogramate utilizabile în numeroase aplicații. Flexibilitatea microprogramării acestui circuit permite emularea eficientă a majorității mașinilor de calcul digitale [1].

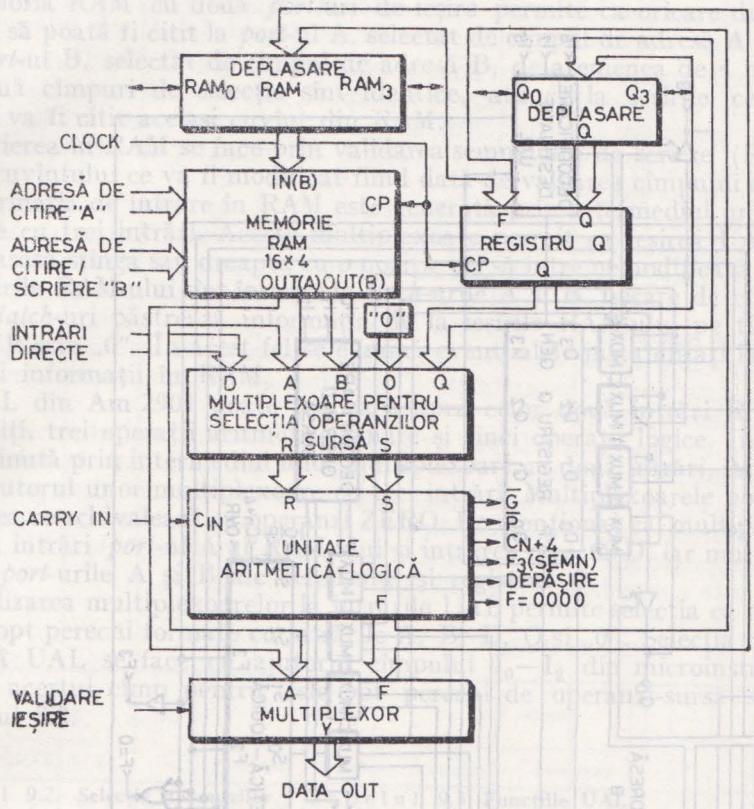


Fig. 9.1. Schema-bloc a microprocesorului *bit-slice* Am 2901

Microprocesorul, așa cum se vede în schema-bloc din figura 9.1, este compus din următoarele elemente principale:

— o memorie RAM de 16 × 4 biți cu două *port*-uri de ieșire;

— o unitate aritmetică-logică de mare viteză;

— un registru auxiliar Q;

— circuite pentru deplasări, decodificări și multiplexări.

Cuvîntul de comandă al microprocesorului, microinstrucțiunea, are nouă biți organizați în trei cîmpuri de cîte trei biți fiecare. Aceste cîmpuri selectează operanzii-sursă pentru UAL, funcția UAL și destinația rezultatului UAL. Microprocesorul are ieșiri „trei-stări", generează diverse semnale de stare din UAL și pote fi conectat în cascadă cu transport-serie sau anticipat.

În figura 9.2 se dă schema detaliată a microprocesorului Am 2901.

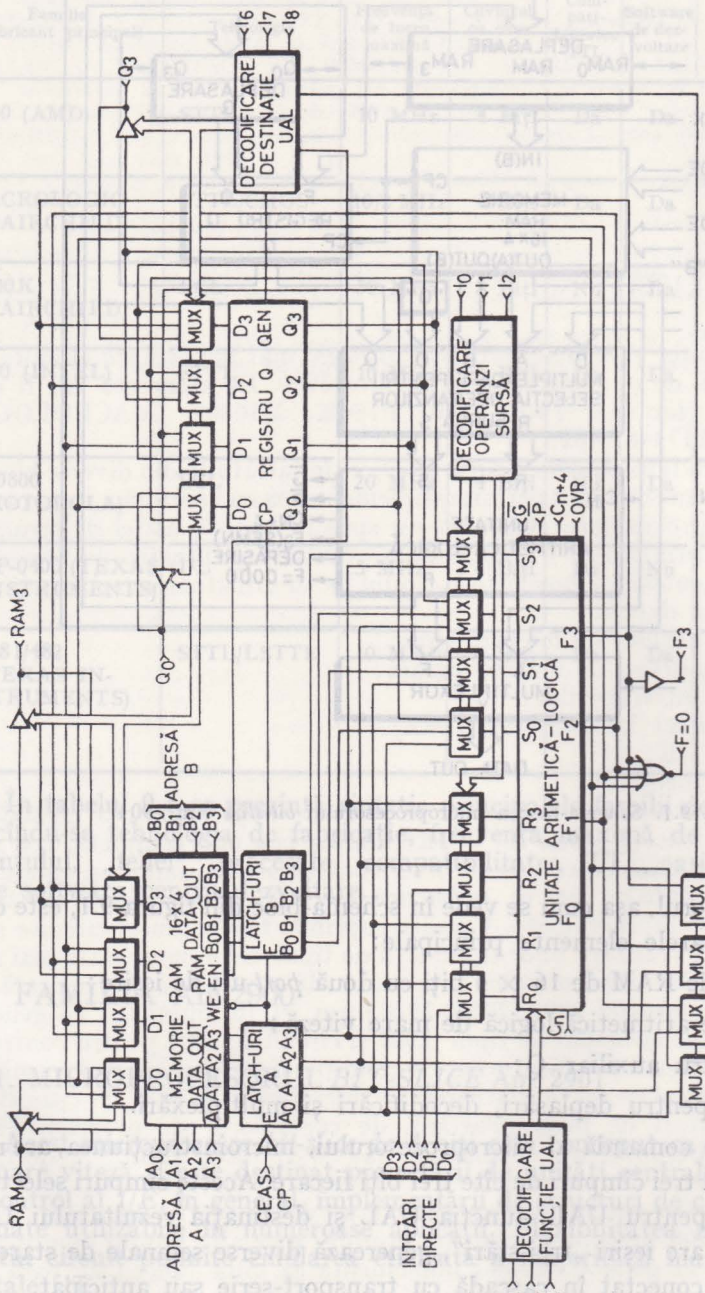


Fig. 9.2. Schema detaliată a microprocesorului Am 2901

Memoria RAM cu două *port*-uri de ieșire permite ca oricare din cele 16 cuvinte să poată fi citit la *port*-ul A, selectat de cîmpul de adresă A de 4 biți, și la *port*-ul B, selectat de cîmpul de adresă B, de asemenea de 4 biți. Dacă cele două cîmpuri de selecție sînt identice, atunci la ieșirile celor două *port*-uri va fi citit același cuvînt din RAM.

Scrierea în RAM se face prin validarea semnalului de scriere (RAMEN), adresa cuvîntului ce va fi modificat fiind dată de valoarea cîmpului de adresă B. Informația de intrare în RAM este generată prin intermediul unor multiplexoare cu trei intrări. Aceste multiplexoare permit ca ieșirea UAL, F, să fie deplasată stînga sau dreapta cu o poziție ori să intre nemodificată în RAM.

Ieșirile RAM-ului sînt înscrise în *latch*-urile A și B, fiecare de cîte 4 biți. Aceste *latch*-uri păstrează informația de la ieșirile RAM-ului pe timpul cît ceasul CP este „0". În acest fel se elimină eventualele instabilități la scrierea unei noi informații în RAM.

UAL din Am 2901 poate realiza asupra celor două intrări R și S, de cîte 4 biți, trei operații aritmetice binare și cinci operații logice. Intrarea R este obținută prin intermediul unor multiplexoare cu două intrări, iar intrarea S, cu ajutorul unor multiplexoare cu trei intrări. Multiplexoarele pot fi inhibate, ceea ce echivalează cu operanzi ZERO. De menționat că multiplexoarele R au ca intrări *port*-ul A al RAM-ului și intrarea directă D, iar multiplexoarele S, *port*-urile A și B ale RAM-ului și registrul Q.

Utilizarea multiplexoarelor la intrările UAL permite selecția ca operanzi-sursă a opt perechi formate cu intrările A, B, D, Q și „0". Selecția operanzilor sursă UAL se face cu ajutorul cîmpului I_0—I_2 din microinstrucțiune. Valorile acestui cîmp pentru cele opt perechi de operanzi-sursă sînt date în tabelul 9.2.

Tabelul 9.2. **Selecția operanzilor**

Cod mnemonic	I_2	I_1	I_0	Operanzii UAL (R, S)	
AQ	0	0	0	A	Q
AB	0	0	1	A	B
ZQ	0	1	0	0	Q
ZB	0	1	1	0	B
ZA	1	0	0	0	A
DA	1	0	1	D	A
DQ	1	1	0	D	Q
DZ	1	1	1	D	0

Tabelul 9.3. **Funcțiile UAL**

Cod mnemonic	I_5	I_4	I_3	Funcția UAL	Simbol
ADD	0	0	0	R Plus S	R+S
SUBR	0	0	1	S Minus R	S−R
SUBS	0	1	0	R Minus S	R−S
OR	0	1	1	R OR S	R∨S
AND	1	0	0	R AND S	R∨S
NOTRS	1	0	1	$\overline{R}$ AND S	$\overline{R}\wedge S$
EXOR	1	1	0	R EX-OR S	R∀S
EXNOR	1	1	1	R EX-NOR S	$\overline{R \forall S}$

Intrarea D reprezintă o intrare directă de informație, utilizată pentru a introduce noi valori în registrele de lucru sau, prin intermediul UAL, pentru a modifica valorile acestor registre. Registrul Q este un registru, tot de 4 biți, destinat în principal implementării rutinelor de înmulțire și împărțire. El poate fi însă utilizat și ca registru acumulator sau de memorare în cazul altor aplicații.

Unitatea aritmetică-logică UAL este un dispozitiv aritmetic și logic de mare viteză ale cărui funcții sînt selectate de cîmpul I_3—I_5 din microinstrucțiune. Valorile acestui cîmp sînt date în tabelul 9.3. Ieșirile $\overline{G}$ și $\overline{P}$, de generare-transport și, respectiv, de propagare-transport, se utilizează ca intrări în generatorul de transport anticipat Am 2902, în cazul unei conectări în cascadă. Ieșirea C_{n+4}, transportul, poate fi utilizată ca indicator de depășire într-un registru de stare-program.

UAL mai are încă trei ieșiri, F3, F=0, OVR, ce pot fi utilizate ca indicatori de condiție. Ieșirea F3 reprezintă cel mai semnificativ bit al UAL, semnul. Ea poate fi utilizată pentru a determina semnul rezultatului fără a valida ieșirile „trei-stări". Ieșirea F=0 este întrebuințată pentru detecția de zero. Este o ieșire de tip „colector în gol" care permite cablarea unui SAU LOGIC între mai multe circuite Am 2901. Ieșirea OVR este folosită pentru a indica, în cazul operațiilor aritmetice, depășirea gamei de numere reprezentate în complement față de doi.

Ieșirile F_0—F_3 ale UAL pot avea diverse destinații selectate de cîmpul I_6—I_8 din microinstrucțiune. Valorile acestui cîmp sînt date în tabelul 9.4. Ieșirile Y_0—Y_3 ale microprocesorului sînt de tip „trei-stări" și se pot conecta direct în sistem magistrală. Multiplexorul cu două intrări care generează aceste ieșiri poate selecta *port*-ul A al RAM-ului sau ieșirile F ale UAL. Această selectare este controlată, de asemenea, de cîmpul I_6—I_8.

Tabelul 9.4. **Destinațiile UAL**

Cod mnemonic	I_8	I_7	I_6	Funcția RAM		Funcția Q		Ieșirea Y	Deplasare RAM		Deplasare Q	
				deplasare	încărcare	deplasare	încărcare		RAM_0	RAM_3	Q_0	Q_3
QREG	0	0	0	X	Nu	Nu	F→Q	F	X	X	X	X
NOP	0	0	1	X	Nu	X	Nu	F	X	X	X	X
RAMA	0	1	0	Nu	F→B	X	Nu	A	X	X	X	X
RAMF	0	1	1	Nu	F→B	X	Nu	F	X	X	X	X
RAMQD	1	0	0	Dreapta	F/2→B	Dreapta	Q/2→Q	F	F_0	IN_3	Q_0	IN_3
RAMD	1	0	1	Dreapta	F/2→B	X	Nu	F	F_0	IN_3	Q_0	X
RAMQU	1	1	0	Stînga	2F→B	Stînga	2Q→Q	F	IN_0	F_3	IN_0	Q_3
RAMU	1	1	1	Stînga	2F→B	X	Nu	F	IN_0	F_3	X	Q_3

Notă: X — fără importanță, intrarea de deplasare este conectată la o ieșire internă în starea a treia; B — registru RAM adresat de intrările B; IN—intrare.

Tabelul 9.5. Operațiile executate de UAL

I_{543}	Funcția UAL	I_{210}							
		0	1	2	3	4	5	6	7
		Sursa UAL							
		A, Q	A, B	0, Q	0, B	0, A	D, A	D, Q	D, 0
0	$C_n = 0$ R Plus S $C_n = 1$	A + Q A + Q + 1	A + B A + B + 1	Q Q + 1	B B + 1	A A + 1	D + A D + A + 1	D + Q D + Q + 1	D D + 1
1	$C_n = 0$ S Minus R $C_n = 1$	Q − A − 1 Q − A	B − A − 1 B − A	Q − 1 Q	B − 1 B	A − 1 A	A − D − 1 A − D	Q − D − 1 Q − D	− D − 1 − D
2	$C_n = 0$ R Minus S $C_n = 1$	A − Q − 1 A − Q	A − B − 1 A − B	− Q − 1 − Q	− B − 1 − B	− A − 1 − A	D − A − 1 D − A	D − Q − 1 D − Q	D − 1 D
3	R OR S	A ∨ Q	A ∨ B	Q	B	A	D ∨ A	D ∨ Q	D
4	R AND S	A ∧ Q	A ∧ B	0	0	0	D ∧ A	D ∧ Q	0
5	$\overline{R}$ AND S	$\overline{A}$ ∧ Q	$\overline{A}$ ∧ B	Q	B	A	$\overline{D}$ ∧ A	$\overline{D}$ ∧ Q	0
6	R EX-OR S	A ∀ Q	A ∀ B	Q	B	A	D ∀ A	D ∀ Q	D
7	R EX-NOR S	$\overline{A \forall Q}$	$\overline{A \forall B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \forall A}$	$\overline{D \forall A}$	$\overline{D}$

Cîmpul I_6—I_8 controlează și operațiile de deplasare. Așa cum s-a spus mai sus, intrarea în RAM se face prin intermediul unor multiplexoare cu trei intrări. Acestea permit ca ieșirile UAL să intre în RAM deplasate dreapta sau stînga. Deplasarea se face prin intermediul a două intrări/ieșiri numite RAM_0 și RAM_3. Astfel, la deplasare stînga se validează ieșirea RAM_3 și intrarea RAM_0. La deplasare dreapta se validează ieșirea RAM_0 și intrarea RAM_3, iar cînd nu se face nici o deplasare intrările/ieșirile RAM_0, RAM_3 sînt invalidate. Cîmpul I_6—I_8 controlează, de asemenea, și unitatea de deplasare asociată registrului Q. Ca și deplasarea RAM-ului deplasarea registrului Q se face prin intermediul unui multiplexor și a unor intrări/ieșiri notate Q_0 și Q_3.

În tabelul 9.5 se sintetizează operațiile executate de UAL asupra tuturor perechilor de operanzi ce pot fi selectate cu ajutorul cîmpului I_0—I_2. Este precizat, de asemenea, și rolul pe care îl are intrarea C_n asupra operațiilor aritmetice. Matricea din această figură definește complet funcționarea UAL împreună cu selecția corespunzătoare a operanzilor.

9.2.2. GENERATORUL DE TRANSPORT ANTICIPAT Am 2902

Am 2902 este un generator de transport anticipat de mare viteză destinat conectării în cascadă a maximum patru UAL binare, acceptînd ca intrări perechi de semnale de tip propagare și generare ($\overline{P}$ și $\overline{G}$). Circuitul are, de asemenea, două ieșiri de tip $\overline{P}$ și $\overline{G}$ care permit utilizarea lui pentru obținerea transportului anticipat în structuri cu mai mult de patru microprocesoare *bit-slice*.

Ecuațiile logice ale funcțiilor generate la ieșirile acestui circuit (fig. 9.3) sînt:

$$C_{n+x} = G_0 + P_0C_n;$$
$$C_{n+y} = G_1 + P_1G_0 + P_1P_0C_n;$$
$$C_{n+z} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_n;$$
$$G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0;$$
$$P = P_3P_2P_1P_0.$$

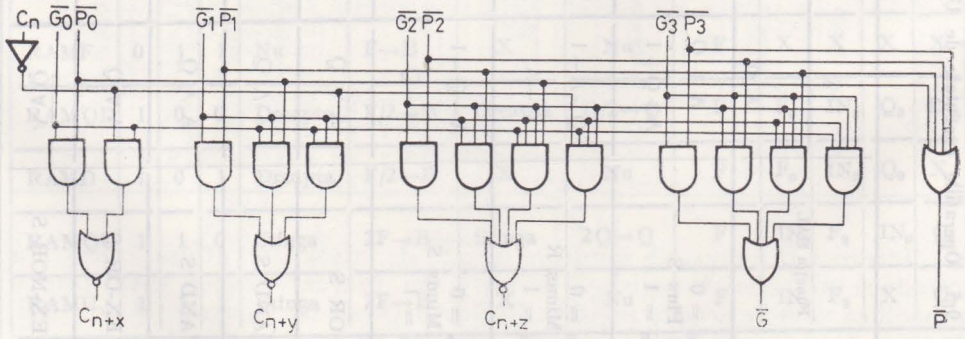


Fig. 9.3. Schema generatorului de transport anticipat Am 2902

9.2.3. MICROPROCESOARELE *BIT-SLICE* Am 2903/29203

Microprocesorul *bit-slice* Am 2903 este un microprocesor pe patru biți expandabil. El realizează toate funcțiunile lui Am 2901, avînd în plus unele îmbunătățiri destinate în special implementării de procesoare aritmetice. Am 29203, similar cu Am 2903, are în plus față de acesta cîteva facilități de I/E, un set de instrucțiuni mai bogat și este cu aproximativ 30 % mai rapid.

Realizate în tehnologie *Advanced Low Power Schottky*, Am 2903/29203 sînt microprocesoare *bit-slice* de înaltă performanță ce pot fi utilizate, ca și Am 2901, în proiectarea unor diverse structuri de control microprogramate.

În figura 9.4 se dă schema-bloc a microprocesoarelor Am 2903/29203. Aceste circuite constau în principal dintr-un RAM de 16 × 4 biți cu două *port*-uri de ieșire și *latch*-urile corespunzătoare, o UAL și o unitate de deplasare asociată ieșirii UAL, un registru Q împreună cu unitatea pentru deplasare asociată lui și un decodificator de instrucțiuni. Cuvîntul de comandă al microprocesorului are 9 biți, I_0—I_8.

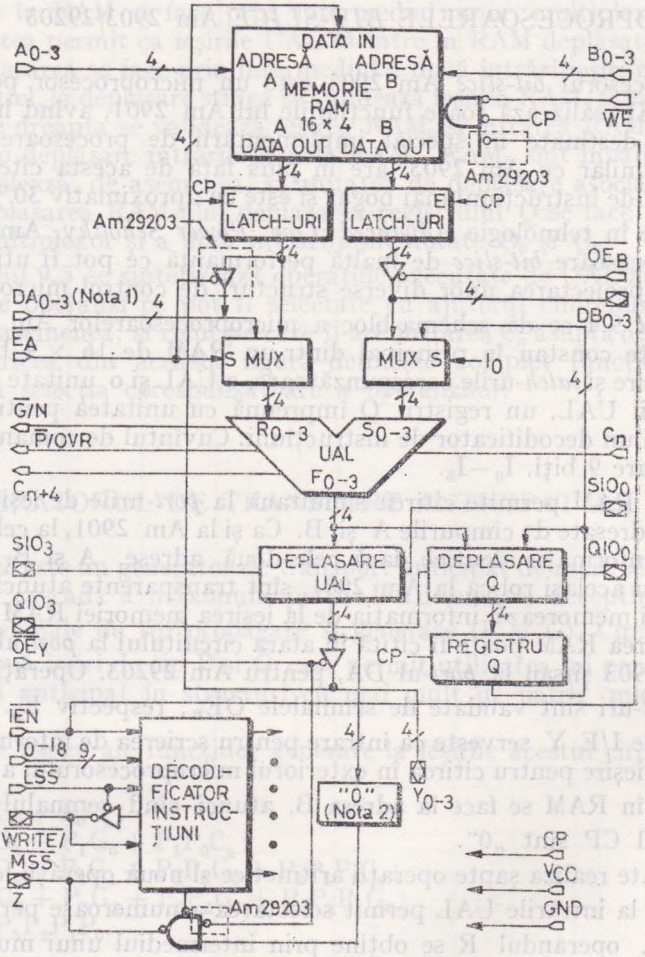
Memoria RAM permite citirea simultană la *port*-urile de ieșire a oricăror două locații adresate de cîmpurile A și B. Ca și la Am 2901, la cele două ieșiri pot fi citite informații identice dacă cele două adrese, A și B, sînt egale. *Latch*-urile, cu același rol ca la Am 2901, sînt transparente atunci cînd ceasul CP este „1" și memorează informația de la ieșirea memoriei RAM, cînd ceasul este „0". Ieșirea RAM poate fi citită în afara circuitului la *port*-ul de I/E DB pentru Am 2903 și/sau la *port*-ul DA, pentru Am 29203. Operațiile de citire la aceste *port*-uri sînt validate de semnalele $\overline{OE}_B$, respectiv $\overline{E}_A$.

Port-ul de I/E Y servește ca intrare pentru scrierea de informații externe în RAM și ca ieșire pentru citirea în exteriorul microprocesorului a ieșirii UAL.

Scrierea în RAM se face la adresa B, atunci cînd semnalul de validare $\overline{WE}$ și ceasul CP sînt „0".

UAL poate realiza șapte operații aritmetice și nouă operații logice. Multiplexoarele de la intrările UAL permit selectarea a numeroase perechi de operanzi. Astfel, operandul R se obține prin intermediul unui multiplexor cu ajutorul semnalului $\overline{E}_A$ care selectează fie intrarea externă DA, fie *port*-ul A al RAM-ului. Operandul S, obținut de asemenea prin intermediul unui multiplexor, poate fi *port*-ul B al RAM-ului, intrarea externă DB sau registrul Q. Selecția se face cu ajutorul semnalelor $\overline{OE}_B$ și I_0. În tabelul 9.6 sînt date toate perechile de operanzi UAL în funcție de semnalele de selecție $\overline{E}_A$, I_0, $\overline{OE}_B$.

Funcțiile realizate de UAL din Am 2903 sînt selectate cu ajutorul biților de instrucțiune I_8—I_0 (tabelele 9.7 și 9.8). Atunci cînd I_4, I_3, I_2, I_1 și I_0 sînt „0" Am 2903 execută funcții speciale (tabelul 9.8) selectate cu ajutorul biților I_8—I_5. Cînd Am 2903 nu efectuează funcții speciale, atunci operația UAL este determinată de valoarea biților I_4—I_1 (tabelul 9.7). UAL din Am 29203 este similară, cu singura diferență că ea execută 16 funcții speciale în loc de 9 cîte execută cea din Am 2903.



Note: 1. DA0–3 este intrare pentru Am 2903 și I/E pentru Am 29203

2. La Am 2903 "0" este conectat la ieșirile Y după buffer-ul $\overline{OE}_Y$

Fig. 9.4. Schema-bloc a microprocesoarelor Am 2903/29203

De remarcat că Am 2903/29203 pot fi conectate în cascadă cu transport-serie sau anticipat. Într-o conectare în cascadă fiecare microprocesor trebuie programat în funcție de poziția lui: cel mai semnificativ, intermediar sau cel mai puțin semnificativ. La conectarea în cascadă se folosesc semnalele $\overline{G}$ și $\overline{P}$ generate de circuitul cel mai puțin semnificativ și de circuitele intermediare.

Tabelul 9.6. **Operanzii UAL**

$\overline{E}_A$	I_0	$\overline{OE}_B$	Operand R	Operand S
0	0	0	Ieșirea „A" a RAM	Ieșirea „B" a RAM
0	0	1	Ieșirea „A" a RAM	DB_{0-3}
0	1	X	Ieșirea „A" a RAM	Registrul Q
1	0	0	DA_{0-3}	Ieșirea „B" a RAM
1	0	1	DA_{0-3}	DB_{0-3}
1	1	X	DA_{0-3}	Registrul Q

Tabelul 9.7. **Funcțiile UAL**

I_4	I_3	I_2	I_1	I_0	Funcții UAL
0	0	0	0	0	Funcții speciale (v. tab. 9.8.)
0	0	0	0	1	$F_i = 1$
0	0	0	1	X	F = S Minus R Minus 1 Plus C_n
0	0	1	0	X	F = R Minus S Minus 1 Plus C_n
0	0	1	1	X	F = R Plus S Plus C_n
0	1	0	0	X	F = S Plus C_n
0	1	0	1	X	$F = \overline{S}$ Plus C_n
0	1	1	0	0	Funcții speciale rezervate
0	1	1	0	1	F = R Plus C_n
0	1	1	1	0	Funcții speciale rezervate
0	1	1	1	1	$F = \overline{R}$ Plus C_n
1	0	0	0	0	Funcții speciale rezervate
1	0	0	0	1	$F_i = 0$
1	0	0	1	X	$F_i = R_i$ AND S_i
1	0	1	0	X	$F_i = R_i$ EXCLUSIVE NOR S_i
1	0	1	1	X	$F_i = R_i$ EXCLUSIVE OR S_i
1	1	0	0	X	$F_i = R_i$ AND S_i
1	1	0	1	X	$F_i = R_i$ NOR S_i
1	1	1	0	X	$F_i = R_i$ NAND S_i
1	1	1	1	X	$F_i = R_i$ OR S_i

Tabelul 9.8. **Funcțiile speciale realizate de Am 2903/29203**

I_8	I_7	I_6	I_5	Funcția specială	Funcția UAL	Deplasarea UAL	Deplasarea Q	Circuitul care implementează funcția
0	0	0	0	Înmulțire fără semn	$F=S+C_n$ dacă $Z=0$ $F=R+S+C_n$ dacă $Z=1$	Deplasare logică $F/2 \rightarrow Y$ (Nota 1)	Deplasare logică $Q/2 \rightarrow Q$	Am 2903 Am 29203
0	0	0	1					Am 29203
0	0	1	0	Înmulțire în complement față de 2	$F=S+C_n$ dacă $Z=0$ $F=R+S+C_n$ dacă $Z=1$	Deplasare logică $F/2 \rightarrow Y$ (Nota 2)	Deplasare logică $Q/2 \rightarrow Q$	Am 2903 Am 29203
0	0	1	1					Am 29203
0	1	0	0	Incrementare cu 1 sau 2	$F=S+1+C_n$	$F \rightarrow Y$	Nu	Am 2903 Am 29203
0	1	0	1	Semn-mărime/complement față de 2	$F=S+C_n$ dacă $Z=0$ $F=\overline{S}+C_n$ dacă $Z=0$	$F \rightarrow Y$ (Nota 3)	Nu	Am 2903 Am 29203
0	1	1	0	Înmulțire în complement față de 2, ultimul ciclu	$F=S+C_n$ dacă $Z=1$ $F=S-R-1+C_n$ dacă $Z=1$	$F/2 \rightarrow Y$ (Nota 2)	Deplasare logică $Q/2 \rightarrow Q$	Am 2903 Am 29203
0	1	1	1					Am 29203
1	0	0	0	Normalizare precizie simplă	$F=S+C_n$	$F \rightarrow Y$	Deplasare logică $2Q \rightarrow Q$	Am 2903 Am 29203
1	0	0	1					Am 29203
1	0	1	0	Normalizare precizie dublă și împărțire-primul ciclu	$F=S+C_n$	Deplasare logică $2F \rightarrow Y$	Deplasare logică $2Q \rightarrow Q$	Am 2903 Am 29203
1	0	1	1					Am 29203
1	1	0	0	Împărțire în complement față de 2	$F=S+R+C_n$ dacă $Z=0$ $F=S-R-1+C_n$ dacă $Z=1$	Deplasare logică $2F \rightarrow Y$	Deplasare logică $2Q \rightarrow Q$	Am 2903 Am 29203
1	1	0	1					Am 29203
1	1	1	0	Împărțire în complement față de 2, corecție și rest	$F=S+R+C_n$ dacă $Z=0$ $F=S-R-1+C_n$ dacă $Z=1$	$F \rightarrow Y$	Deplasare logică $2Q \rightarrow Q$	Am 2903 Am 29203
1	1	1	1					Am 29203

Note:
1. La cel mai semnificativ circuit C_{n+4} este conectat intern la ieșirea Y_2.
2. La cel mai semnificativ circuit $F_3 \forall OVR$ este conectat intern la ieșirea Y_3.
3. La ieșirea Y_3 a celui mai semnificativ circuit se generează $S_3 \forall F_3$.

În afara semnalelor de transport anticipat $\overline{G}$ și $\overline{P}$ microprocesoarele Am 2903/29203 mai generează: semnalul de depășire binară C_{n+4}, indicatorul N (cel mai semnificativ bit al ieșirii UAL) care se poate utiliza pentru determinarea semnului, semnalul de depășire OVR utilizat pentru a indica operațiile aritmetice care depășesc gama de numere reprezentate în complement față de doi. Pentru generarea semnalelor $\overline{G}$, $\overline{P}$, N, OVR se folosesc numai două ieșiri, astfel încît semnalele $\overline{G}$ și $\overline{P}$ sînt obținute la ieșirile celui mai puțin semnificativ circuit și ale circuitelor intermediare, iar semnalele N și OVR — la ieșirile celui mai semnificativ circuit.

Microprocesoarele Am 2903/29203 au prevăzută o intrare/ieșire de zero. Această intrare/ieșire, Z, de tip „colector în gol", poate fi conectată într-un SAU LOGIC între mai multe circuite. Ca ieșire se poate utiliza drept indicator de zero, semnalînd situația în care pinii Y_0—Y_3 sînt „0". Pentru Am 29203 ieșirea Z poate fi „1" numai dacă semnalul de validare $\overline{OE}_Y$ este „1". În acest fel detecția de zero se poate face pe mai puțin de un cuvînt.

În tabelul 9.9 sînt date ieșirile UAL, Y_0—Y_3, în funcție de instrucțiunea executată.

Unitatea de deplasare UAL permite trecerea nemodificată a ieșirii F a UAL, deplasarea stînga cu o poziție (2F) și deplasarea dreapta cu o poziție (F/2). Sînt posibile atît deplasări aritmetice, cît și deplasări logice. Deplasarea se face, ca și la Am 2901, prin intermediul a două intrări/ieșiri notate SIO_0 și SIO_3. În timpul unei deplasări stînga SIO_0 este validată ca intrare, iar SIO_3 ca ieșire. În timpul unei deplasări dreapta SIO_3 este validată ca intrare și SIO_0 ca ieșire. În tabelul 9.9 se dau și semnificațiile semnalelor SIO_0 și SIO_3 în funcție de instrucțiunea executată.

În unitatea de deplasare există și un generator/controlor de paritate pe cinci biți care permite detecția erorilor la ieșirea UAL. Paritatea poate fi generată, sub controlul instrucțiunii, la ieșirea SIO_0 pentru F_0, F_1, F_2, F_3, SIO_3. În tabelele 9.8 și 9.9 sînt definite toate operațiile executate de unitatea de deplasare asociată UAL.

Registrul auxiliar Q este destinat, ca și la Am 2901, în principal implementării rutinelor de înmulțire și împărțire putînd fi însă utilizat, în unele aplicații, și ca registru acumulator sau de memorare. Acest registru poate fi selectat ca operand-sursă pentru UAL și încărcat cu ieșirea F a UAL. Unitatea de deplasare asociată registrului Q asigură numai deplasări logice stînga (2Q) sau dreapta (Q/2). Intrările/ieșirile folosite pentru aceste deplasări sînt QIO_0 și QIO_3.

Am 2903/29203 permit executarea operațiilor de deplasare logice și aritmetice pe dublu-cuvînt, prin conectarea ieșirii QIO_3 a celui mai semnificativ circuit la intrarea SIO_0 a celui mai puțin semnificativ și prin executarea unei instrucțiuni de deplasare atît a ieșirii UAL, cît și a registrului Q. Operațiile executate de registrul Q și de unitatea de deplasare asociată lui, în funcție de biții de instrucțiune I_8—I_5, sînt date, de asemenea, în tabelele 9.8 și 9.9.

Semnalele de control interne sînt generate de un decodificator de instrucțiuni în funcție de: intrările de instrucțiune I_0—I_8, intrarea de validare-instrucțiune $\overline{IEN}$, intrarea $\overline{LSS}$ și intrarea/ieșirea WRITE/MSS.

Tabelul 9.9. Ieșirile UAL în funcție de instrucțiunea executată

I_3	I_2	I_1	I_0	Deplasare UAL	SIO_3		Y_3		Y_2		Y_1	Y_0	SIO_0	Deplasare Q	QIO_3	QIO_0
					C.M.S. circuit	alte circuite	C.M.S. circuit	alte circuite	C.M.S. circuit	alte circuite						
0	0	0	0	F/2→Y(A)	Intrare	Intrare	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	Nu	Hi-Z	Hi-Z
0	0	0	1	F/2→Y(L)	Intrare	Intrare	SIO_3	SIO_3	F_3	F_2	F_2	F_1	F_0	Nu	Hi-Z	Hi-Z
0	0	1	0	F/2→Y(A)	Intrare	Intrare	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	Q/2→Q(L)	Intrare	Q_0
0	0	1	1	F/2→Y(L)	Intrare	Intrare	SIO_3	SIO_3	F_3	F_2	F_2	F_1	F_0	Q/2→Q(L)	Intrare	Q_0
0	1	0	0	F→Y	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	Nu	Hi-Z	Hi-Z
0	1	0	1	F→Y	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	Q/2→Q(L)	Intrare	Q_0
0	1	1	0	F→Y	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	F→Q	Hi-Z	Hi-Z
0	1	1	1	F→Y	Intrare	Intrare	F_3	F_3	F_2	F_2	F_1	F_0	Paritate	F→Q	Hi-Z	Hi-Z
1	0	0	0	2F→Y(A)	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Intrare	Nu	Hi-Z	Hi-Z
1	0	0	1	2F→Y(L)	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Intrare	Nu	Hi-Z	Hi-Z
1	0	1	0	2F→Y(A)	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Intrare	2Q→Q(L)	Q_3	Intrare
1	0	1	1	2F→Y(L)	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Intrare	2Q→Q(L)	Q_3	Intrare
1	1	0	0	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	Nu	Hi-Z	Hi-Z
1	1	0	1	F→Y	F_3	F_2	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	2Q→Q(L)	Q_3	Intrare
1	1	1	0	SIO_0→Y_0—Y_3	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	Intrare	Nu	Hi-Z	Hi-Z
1	1	1	1	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	Nu	Hi-Z	Hi-Z

Notă: A = deplasare aritmetică; L = deplasare logică; Hi-Z = starea a treia; C.M.S = cel mai semnificativ

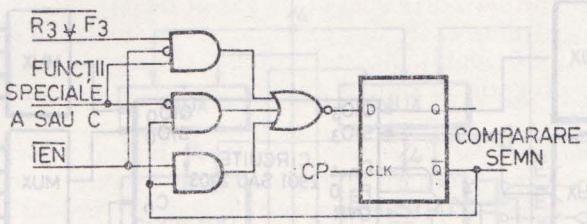


Fig. 9.5. Bistabilul de comparare-semn.

Ieșirea $\overline{\text{WRITE}}$ este „0" atunci cînd se execută o instrucțiune de scriere în RAM. La Am 2903, dacă $\overline{\text{IEN}}$ este „1", ieșirea $\overline{\text{WRITE}}$ este forțată și ea pe „1" reținîndu-se conținutul registrului Q și al bistabilului de comparare-semn (figura 9.5). Dacă $\overline{\text{IEN}}$ este „0" ieșirea $\overline{\text{WRITE}}$ va fi validată, registrul Q și bistabilul de comparare-semn modificîndu-se conform instrucțiunii executate. Bistabilul de comparare-semn este un bistabil intern utilizat în timpul operațiilor de împărțire. Semnalul de comparare-semn apare la ieșirea Z a celui mai semnificativ circuit în timpul funcțiilor speciale C, D, E și F. La Am 29203 $\overline{\text{IEN}}$ controlează scrierea internă fără să afecteze semnalul $\overline{\text{WRITE}}$. Semnalul $\overline{\text{IEN}}$ poate fi în acest fel controlat separat pentru fiecare circuit, facilitîndu-se astfel operațiile pe octet.

Prin punerea intrării $\overline{\text{LSS}}$ la „0" circuitul se programează pentru a funcționa în poziția cea mai puțin semnificativă, validîndu-se semnalul de ieșire $\overline{\text{WRITE}}$ la pinul bidirecțional $\overline{\text{WRITE}}/\overline{\text{MSS}}$. Dacă intrarea $\overline{\text{LSS}}$ este pe „1", pinul $\overline{\text{WRITE}}/\overline{\text{MSS}}$ devine intrare: prin conectarea lui la „1" circuitul va putea funcționa într-o poziție intermediară, iar prin conectarea lui la „0", în poziția cea mai semnificativă.

9.2.4. CIRCUITUL PENTRU CONTROLUL DEPLASĂRII ȘI AL INDICATORILOR DE CONDIȚIE Am 2904

Am 2904 este un circuit destinat realizării unor operații legate de funcționarea unei UAL, implementate de obicei cu ajutorul unor circuite integrate MSI (fig. 9.6). Printre acestea se numără generarea transportului, interconectarea căilor de date, controlul operațiilor de deplasare. Am 2904 realizează aceste funcții cu ajutorul a trei blocuri logice independente. Astfel, transportul se generează cu un multiplexor, iar controlul deplasărilor cu alte patru multiplexoare cu ieșiri de tip „trei-stări". Pentru memorarea indicatorilor de condiție se folosesc două registre. Multiplexorul de testare a condițiilor poate selecta valoarea adevărată sau negată a indicatorilor de condiție, precum și anumite combinații ale lor în scopul implementării unor funcții de test mai complexe.

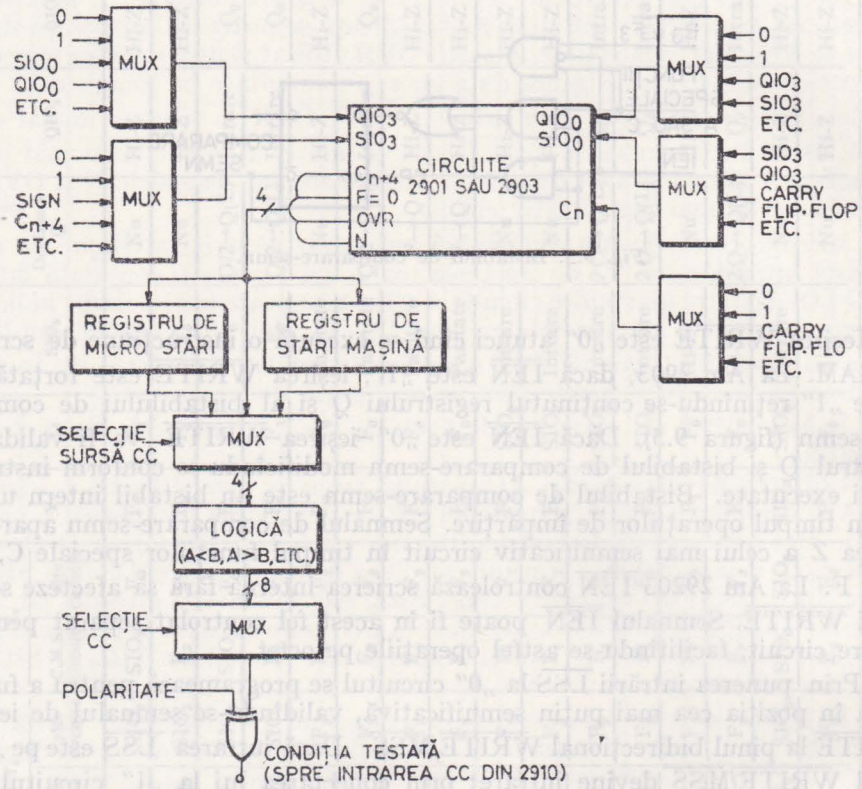


Fig. 9.6. Realizarea cu circuite integrate MSI a funcțiilor lui Am 2904.

9.2.4.1. Structura circuitului

Schema-bloc a circuitului este dată în figura 9.7. Am 2904 permite implementarea a patru funcțiuni necesare tuturor procesoarelor:

— registrul de stare-program;
— testarea indicatorilor de condiție;
— controlul deplasărilor;
— controlul transportului.

Cuvîntul de comandă al circuitului are 13 biți notați I_0—I_{12}.

Registrul de stare-program. Am 2904 conține două registre de cîte patru biți în care se pot memora indicatorii de stare ai unei UAL: transportul (C), semnul (N), indicatorul de zero (Z), depășirea (OVR). Aceste registre sînt numite Registrul de Micro-Stări μSR (Micro Status Register) și Registrul de Stări-Mașină MSR (Machine Status Register). Fiecare din aceste registre poate fi controlat independent. Registrele sînt compuse din bistabili D acționați pe frontul pozitiv al ceasului CP.

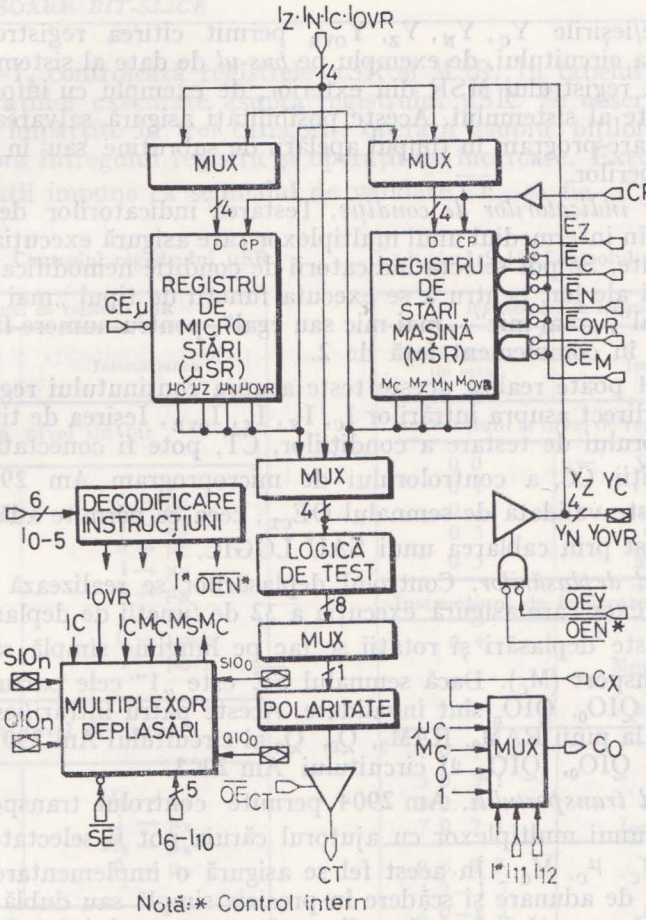


Fig. 9.7. Schema-bloc a circuitului Am 2904

Registrul μSR poate fi încărcat sub controlul biților de instrucțiune I_0—I_5 cu indicatorii de condiție I_C, I_N, I_Z, I_{OVR} sau cu conținutul registrului MSR. De asemenea, sub controlul biților de instrucțiune, biții registrului μSR pot fi modificați individual. Intrarea de validare $\overline{CE}_\mu$ permite inhibarea registrului atunci cînd este pe „1".

Registrul MSR poate fi încărcat, de asemenea, sub controlul biților I_0—I_5, cu indicatorii de condiție I_C, I_N, I_Z, I_{OVR}, cu conținutul registrului μSR și cu valoarea intrărilor/ieșirilor Y_C, Y_N, Y_Z, Y_{OVR}. Registrul MSR mai poate fi șters, poziționat pe „1" sau complementat. Bistabilii registrului pot fi actualizați selectiv controlînd cele patru intrări individuale de validare $\overline{E}_C$, $\overline{E}_N$, $\overline{E}_Z$, $\overline{E}_{OVR}$ și intrarea generală de validare $\overline{CE}_M$. De exemplu, dacă $\overline{CE}_M$ și $\overline{E}_Z$ sînt „0" se validează actualizarea indicatorului Z. Dacă o intrare de validare este pe „1", atunci se inhibă modificarea indicatorului respectiv. Modificarea întregului registru este inhibată dacă $\overline{CE}_M$ este „1".

Intrările/ieșirile Y_C, Y_N, Y_Z, Y_{OVR} permit citirea registrelor μSR și MSR în afara circuitului, de exemplu pe *bus-ul* de date al sistemului, precum și încărcarea registrului MSR din exterior, de exemplu cu informația de pe *bus-ul* de date al sistemului. Aceste posibilități asigură salvarea și refacerea registrului stare-program în timpul apelării de subrutine sau în timpul servirii întreruperilor.

Testarea indicatorilor de condiție. Testarea indicatorilor de condiție se realizează prin intermediul unui multiplexor care asigură execuția a 16 funcții de test diferite. Se pot selecta indicatorii de condiție nemodificați sau negați și combinații ale lor, pentru a se executa funcții de tipul „mai mare", „mai mare sau egal", „mai mic", „mai mic sau egal", pentru numere fără semn sau reprezentate în complement față de 2.

Am 2904 poate realiza aceste teste asupra conținutului registrelor μSR și MSR sau direct asupra intrărilor I_C, I_N, I_Z, I_{OVR}. Ieșirea de tip „trei-stări" a multiplexorului de testare a condițiilor, CT, pote fi conectată la intrarea de test-condiții, $\overline{CC}$, a controlorului de microprogram Am 2910 (§ 9.2.6). Ieșirea CT este validată de semnalul $\overline{OE}_{CT}$, ceea ce permite adăugarea altor intrări de test prin cablarea unui SAU LOGIC.

Controlul deplasărilor. Controlul deplasărilor se realizează cu ajutorul unui multiplexor care asigură execuția a 32 de funcții de deplasare și rotație diferite. Aceste deplasări și rotații se fac pe lungime simplă sau dublă, cu sau fără transport (M_C). Dacă semnalul $\overline{SE}$ este „1" cele patru intrări/ieșiri SIO_0, SIO_n, QIO_0, QIO_n sînt invalidate. Aceste patru intrări/ieșiri se pot conecta direct la pinii RAM_0, RAM_3, Q_0, Q_3 ai circuitului Am 2901 sau la pinii SIO_0, SIO_3, QIO_0, QIO_3 ai circuitului Am 2903.

Controlul transportului. Am 2904 permite controlul transportului prin intermediul unui multiplexor cu ajutorul căruia pot fi selectate intrările 0 1, C_x, μ_C, M_C, $\overline{\mu}_C$, $\overline{M}_C$. În acest fel se asigură o implementare mai ușoară a operațiilor de adunare și scădere în precizie simplă sau dublă. Intrarea C_x este destinată conectării la ieșirea Z a microprocesorului Am 2903 pentru a se facilita execuția unora din instrucțiunile speciale ale microprocesorului. Ieșirea C_0 se poate conecta la pinul C_n al celui mai puțin semnificativ Am 2901 sau Am 2903 și la pinul C_n al generatorului de transport anticipat Am 2902.

9.2.4.2. Instrucțiunile lui Am 2904

Am 2904 este controlat cu ajutorul a 13 biți de instrucțiune I_0—I_{12} și a nouă linii de validare $\overline{CE}_M$, $\overline{CE}_\mu$, $\overline{E}_C$, $\overline{E}_N$, $\overline{E}_Z$, $\overline{E}_{OVR}$, $\overline{OE}_Y$, $\overline{OE}_{CT}$, $\overline{SE}$. În majoritatea structurilor microprogramate se urmărește o economie de memorie PROM prin micșorarea cîmpurilor de control. Acest lucru se poate obține pentru Am 2904 prin conectarea la „0" sau „1" a unora din intrările enumerate mai sus, prin conectarea împreună a unora din ele sau prin decodificarea cîmpului de microinstrucțiune destinat controlului acestui circuit.

Biții I_0—I_5 controlează registrele μSR și MSR. În tabelul 9.10 se dau codurile operațiilor executate asupra registrului μSR. Se observă că aceste operații sînt împărțite în trei categorii: operații asupra biților registrului, operații asupra întregului registru și operații de încărcare. Execuția tuturor acestor operații impune ca semnalul de validare $\overline{CE}_\mu$ să fie „0".

Tabelul 9.10. **Controlul registrului μSR**

Instrucțiuni cu registrul μSR	
$I_5I_4I_3I_2I_1I_0$ (în octal)	Instrucțiunea
Instrucțiuni la nivel de bit	
1 0	$0 \rightarrow \mu_Z$
1 1	$1 \rightarrow \mu_Z$
1 2	$0 \rightarrow \mu_C$
1 3	$1 \rightarrow \mu_C$
1 4	$0 \rightarrow \mu_N$
1 5	$1 \rightarrow \mu_N$
1 6	$0 \rightarrow \mu_{OVR}$
1 7	$1 \rightarrow \mu_{OVR}$
Instrucțiuni la nivel de registru	
0 0	$M_X \rightarrow \mu_X$
0 1	$1 \rightarrow \mu_X$
0 2	$M_X \rightarrow \mu_X$
0 3	$0 \rightarrow \mu_X$
Instrucțiuni de încărcare	
0 6, 0 7	$I_Z \rightarrow \mu_Z$ $I_C \rightarrow \mu_C$ $I_N \rightarrow \mu_N$ $I_{OVR} + \mu_{OVR} \rightarrow \mu_{OVR}$
3 0, 3 1, 5 0, 5 1, 7 0, 7 1	$I_Z \rightarrow \mu_Z$ $\overline{I_C} \rightarrow \mu_C$ $I_N \rightarrow \mu_N$ $I_{OVR} \rightarrow \mu_{OVR}$
0 4, 0 5, 2 0—2 7 3 2—4 7 5 2—6 7 7 2—7 7	$I_Z \rightarrow \mu_Z$ $I_C \rightarrow \mu_C$ $I_N \rightarrow \mu_N$ $I_{OVR} \rightarrow \mu_{OVR}$

Tabelul 9.11. **Controlul registrului MSR**

Instrucțiuni cu registrul MSR	
$I_5I_4I_3I_2I_1I_0$ (în octal)	Instrucțiunea
Instrucțiuni la nivel de registru	
0 0	$Y_X \rightarrow M_X$
0 1	$1 \rightarrow M_X$
0 2	$\mu_X \rightarrow M_X$
0 3	$0 \rightarrow M_X$
0 5	$\overline{M_X} \rightarrow M_X$
Instrucțiuni de încărcare	
0 4	$I_Z \rightarrow M_Z$ $M_{OVR} \rightarrow M_C$ $I_N \rightarrow M_N$ $M_C \rightarrow M_{OVR}$
1 0, 1 1, 3 0, 3 1 5 0, 5 1, 7 0, 7 1	$I_Z \rightarrow M_Z$ $\overline{I_C} \rightarrow M_C$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$
0 6, 0 7, 1 2—1 7, 2 0—2 7, 3 2—3 7 4 0—4 7, 5 2—6 7, 7 2—7 7	$I_Z \rightarrow M_Z$ $I_C \rightarrow M_C$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$

Tabelul 9.12. **Controlul ieșirii Y**

$\overline{OE}_Y$	I_5	I_4	Ieșirea Y
1	X	X	Hi-Z
0	0	X	$\mu_i \rightarrow Y_i$
0	1	0	$M_i \rightarrow Y_i$
0	1	1	$I_i \rightarrow Y_i$

Notă: X — fără importanță;
Hi-Z — starea a treia

În tabelul 9.11 se dau codurile operațiilor executate asupra registrului MSR. Aceste operații se împart și ele în două categorii: operații la nivel de registru și operații de încărcare. Execuția acestor operații este condiționată de poziționarea semnalului $\overline{CE}_M$ pe „0". Operațiile la nivel de bit se realizează prin utilizarea operațiilor la nivel de registru sau a operațiilor de încărcare și prin poziționarea unuia din semnalele de validare $\overline{E}_C$, $\overline{E}_N$, $\overline{E}_Z$, $\overline{E}_{OVR}$.

Biții I_4—I_5 controlează și ieșirea Y a circuitului (tabelul 9.12), iar biții I_0—I_3, ieșirea CT a multiplexorului de testare a indicatorilor de condiție (tabelul 9.13).

Biții I_6—I_{10} controlează multiplexorul pentru deplasări (tabelul 9.14), iar biții I_{11}—I_{12} și I* de la decodificatorul intern de instrucțiune, multiplexorul pentru controlul transportului (tabelul 9.15).

Pentru detalii referitoare la funcționarea circuitului Am 2904 vezi și § 10.2 în care se descrie o schemă de utilizare a circuitului.

Tabelul 9.13. **Ieșirea CT**

I_3	I_2	I_1	I_0	$I_5 = 0$, $I_4 = 0$	$I_5 = 0$, $I_4 = 1$	$I_5 = 1$, $I_4 = 0$	$I_5 = 1$, $I_4 = 1$
0	0	0	0	$(\mu_N \oplus \mu_{OVR}) + \mu_Z$	$(\mu_N \oplus \mu_{OVR}) + \mu_Z$	$(M_N \oplus M_{OVR}) + M_Z$	$(I_N \oplus I_{OVR}) + I_Z$
0	0	0	1	$(\mu_N \odot \mu_{OVR}) \cdot \mu_Z$	$(\mu_N \odot \mu_{OVR}) \cdot \mu_Z$	$(M_N \odot M_{OVR}) + \overline{M}_Z$	$(I_N \odot I_{OVR}) \cdot \overline{I}_Z$
0	0	1	0	$\mu_N \oplus \mu_{OVR}$	$\mu_N \oplus \mu_{OVR}$	$M_N \oplus M_{OVR}$	$I_N \oplus I_{OVR}$
0	0	1	1	$\mu_N \odot \mu_{OVR}$	$\mu_N \odot \mu_{OVR}$	$M_N \odot M_{OVR}$	$I_N \odot I_{OVR}$
0	1	0	0	μ_Z	μ_Z	M_Z	I_Z
0	1	0	1	$\overline{\mu}_Z$	$\overline{\mu}_Z$	$\overline{M}_Z$	$\overline{I}_Z$
0	1	1	0	μ_{OVR}	μ_{OVR}	M_{OVR}	I_{OVR}
0	1	1	1	$\overline{\mu}_{OVR}$	$\overline{\mu}_{OVR}$	$\overline{M}_{OVR}$	$\overline{I}_{OVR}$
1	0	0	0	$\mu_C + \mu_Z$	$\mu_C + \mu_Z$	$M_C + M_Z$	$\overline{I}_C + I_Z$
1	0	0	1	$\overline{\mu}_C \cdot \overline{\mu}_Z$	$\overline{\mu}_C \cdot \overline{\mu}_Z$	$\overline{M}_C \cdot \overline{M}_Z$	$I_C \cdot \overline{I}_Z$
1	0	1	0	μ_C	μ_C	M_C	I_C
1	0	1	1	$\overline{\mu}_C$	$\overline{\mu}_C$	$\overline{M}_C$	$\overline{I}_C$
1	1	0	0	$\overline{\mu}_C + \mu_Z$	$\overline{\mu}_C + \mu_Z$	$\overline{M}_C + M_Z$	$\overline{I}_C + I_Z$
1	1	0	1	$\mu_C \cdot \overline{\mu}_Z$	$\mu_C \cdot \overline{\mu}_Z$	$M_C \cdot \overline{M}_Z$	$I_C \cdot \overline{I}_Z$
1	1	1	0	$I_N \oplus M_N$	μ_N	M_N	I_N
1	1	1	1	$I_N \odot M_N$	$\overline{\mu}_N$	$\overline{M}_N$	$\overline{I}_N$

Notă: $\oplus$ SAU EXCLUSIV
$\odot$ SAU NU EXCLUSIV

Tabelul 9.14. **Operațiile de deplasare**

I_{10}	I_9	I_8	I_7	I_6	M_C RAM Q	SIO_0	SIO_n	QIO_0	QIO_n	Ce se încarcă în M_C
					MSB LSB MSB LSB					
0	0	0	0	0	0 → 0 →	Hi-Z	0	Hi-Z	0	
0	0	0	0	1	1 → 1 →	Hi-Z	1	Hi-Z	1	
0	0	0	1	0	0 → M_N	Hi-Z	0	Hi-Z	M_N	SIO_0
0	0	0	1	1	1 → M_N	Hi-Z	1	Hi-Z	SIO_0	
0	0	1	0	0	→	Hi-Z	M_C	Hi-Z	SIO_0	
0	0	1	0	1	M_N →	Hi-Z	M_N	Hi-Z	SIO_0	
0	0	1	1	0	M_N 0 →	Hi-Z	0	Hi-Z	SIO_0	
0	0	1	1	1	0 →	Hi-Z	0	Hi-Z	SIO_0	QIO_0
0	1	0	0	0	→	Hi-Z	SIO_0	Hi-Z	QIO_0	SIO_0
0	1	0	0	1	→	Hi-Z	M_C	Hi-Z	QIO_0	SIO_0
0	1	0	1	0	→	Hi-Z	SIO_0	Hi-Z	QIO_0	
0	1	0	1	1	I_C →	Hi-Z	I_C	Hi-Z	SIO_0	
0	1	1	0	0	→	Hi-Z	M_C	Hi-Z	SIO_0	QIO_0
0	1	1	0	1	→	Hi-Z	QIO_0	Hi-Z	SIO_0	QIO_0
0	1	1	1	0	$I_N \oplus I_{OVR}$ →	Hi-Z	$I_N \oplus I_{OVR}$	Hi-Z	SIO_0	
0	1	1	1	1	→	Hi-Z	QIO_0	Hi-Z	SIO_0	
					MSB LSB MSB LSB					
1	0	0	0	0	→ 0 → 0	0	Hi-Z	0	Hi-Z	SIO_n
1	0	0	0	1	→ 1 → 1	1	Hi-Z	1	Hi-Z	SIO_n
1	0	0	1	0	→ 0 → 0	0	Hi-Z	0	Hi-Z	
1	0	0	1	1	→ 1 → 1	1	Hi-Z	1	Hi-Z	
1	0	1	0	0	→ 0	QIO_n	Hi-Z	0	Hi-Z	SIO_n
1	0	1	0	1	→ 1	QIO_n	Hi-Z	1	Hi-Z	SIO_n
1	0	1	1	0	→ 0	QIO_n	Hi-Z	0	Hi-Z	
1	0	1	1	1	→ 1	QIO_n	Hi-Z	1	Hi-Z	
1	1	0	0	0	→	SIO_n	Hi-Z	QIO_n	Hi-Z	SIO_n
1	1	0	0	1	→	M_C	Hi-Z	QIO_n	Hi-Z	SIO_n
1	1	0	1	0	→	SIO_n	Hi-Z	QIO_n	Hi-Z	
1	1	0	1	1	→ 0	M_C	Hi-Z	0	Hi-Z	
1	1	1	0	0	→	QIO_n	Hi-Z	M_C	Hi-Z	SIO_n
1	1	1	0	1	→	QIO_n	Hi-Z	SIO_n	Hi-Z	SIO_n
1	1	1	1	0	→	QIO_n	Hi-Z	M_C	Hi-Z	
1	1	1	1	1	→	QIO_n	Hi-Z	SIO_n	Hi-Z	

Notă: Hi-Z — starea a treia

Tabelul 9.15 **Controlul transportului**

I_{12}	I_{11}	I_5	I_3	I_2	I_1	C_0
0	0	X	X	X	X	0
0	1	X	X	X	X	1
1	0	X	X	X	X	C_X
1	1	0	0	X	X	μ_C
1	1	0	X	1	X	μ_C
1	1	0	X	X	1	μ_C
1	1	0	1	0	0	$\overline{\mu_C}$
1	1	1	0	X	X	M_C
1	1	1	X	1	X	M_C
1	1	1	X	X	1	M_C
1	1	1	1	0	0	$\overline{M_C}$

9.2.5. SECVENȚIATOARELE DE MICROPROGRAM Am 2909/2911 ȘI CIRCUITUL DE CONTROL Am 29811

9.2.5.1. Structura secvențiatoarelor Am 2909/2911

Am 2909/2911 sînt controloare de adresă de 4 biți destinate controlului adresei de microprogram, deci generării adresei pentru memoria de microprograme. Circuitul poate fi conectat în cascadă pentru a se obține adrese de 4, 8, 12, 16, ..., biți.

Schema-bloc a circuitelor este dată în figura 9.8, iar schema detaliată în figura 9.9.

Adresa de microprogram este selectată prin intermediul unui multiplexor între intrările directe D, intrările AR memorate într-un registru intern, ieșirea F a unei stive LIFO* de 4 cuvinte și numărătorul de microprogram μPC. Acest multiplexor de selecție este controlat de semnalele S_1 și S_0 (tabelul 9.16*a*).

Registrul intern de adresă, AR, constă din 4 bistabili de tip D acționați pe frontul pozitiv al ceasului și validați de semnalul $\overline{RE}$. Cînd acest semnal este „0" în registrul intern se poate înscrie o nouă informație.

* Last In — First Out

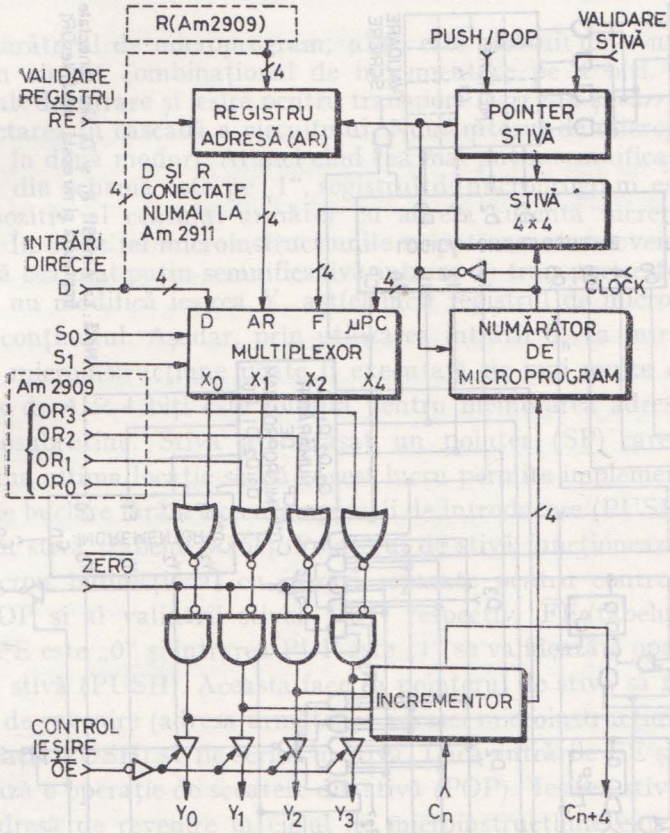


Fig. 9.8. Schema-bloc a secvenţiatoarelor Am 2909/2911

Tabelul 9.16*a*.

Selecţia adresei

S_1	S_0	Informaţia la ieşirile Y_i
0	0	Numărătorul de microprogram (μPC)
0	1	Registrul de adresă (AR)
1	0	Stiva de subrutine (STKO)
1	1	Intrările directe (D_i)

Tabelul 9.16*b*.

Operaţiile cu stiva

$\overline{FE}$	PUP	Acţiunea asupra stivei
1	X	Nu se modifică
0	1	Se incrementează pointerul, se introduce PC în stivă (PUSH)
0	0	Se scoate din stivă (POP)

Tabelul 9.16c.

Ieşirile Y_i

OR_i	$\overline{ZERO}$	$\overline{OE}$	Ieşirile Y_i
X	X	1	Hi-Z
X	0	0	0
1	1	0	1
0	1	0	Selectate de S_1 S_0

Notă: Hi-Z — starea a treia

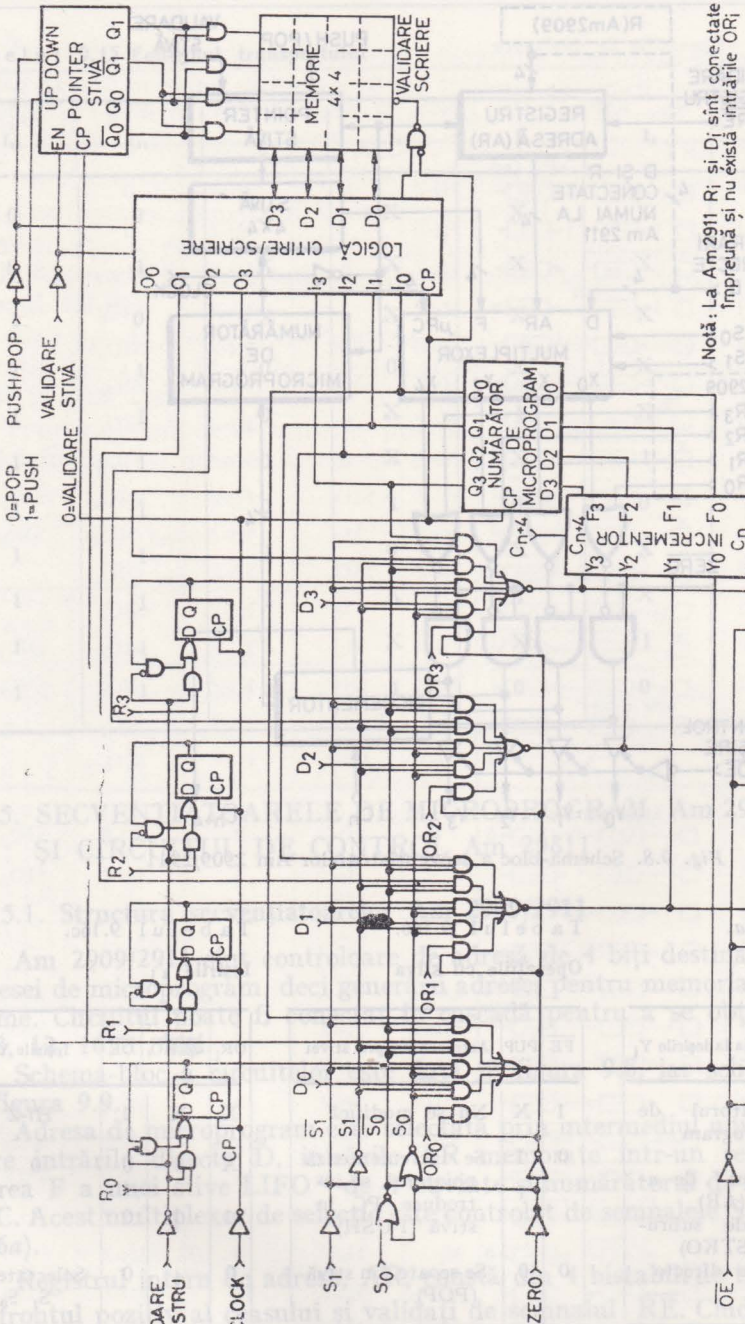


Fig. 9.9. Schema detaliată a secvențiatorilor Am 2909/2911

Numărătorul de microprogram, μPC, este alcătuit dintr-un registru de 4 biți și un circuit combinațional de incrementare pe 4 biți. Incrementorul are semnale de intrare și ieșire pentru transport (*carry-in* și *carry-out*) care permit conectarea în cascadă a circuitului. Numărătorul de microprogram poate fi utilizat în două moduri. Atunci cînd cea mai puțin semnificativă intrare de transport din schemă este pe „1", registrul de microprogram este încărcat pe frontul pozitiv al ceasului următor cu adresa curentă incrementată cu 1, (Y + 1). În acest fel microinstrucțiunile se pot executa secvențial, una după alta. Dacă cea mai puțin semnificativă intrare de transport este pe „0", incrementorul nu modifică ieșirea Y, astfel încît registrul de microprogram nu-și schimbă conținutul. Așadar, prin utilizarea intrării C_n ca intrare de control o aceeași microinstrucțiune poate fi executată de mai multe ori.

Stiva de 4 × 4 biți este folosită pentru memorarea adresei de revenire din microsubrutine. Stiva are atașat un pointer (SP) care „punctează" întotdeauna ultima locație scrisă. Acest lucru permite implementarea instrucțiunilor de buclare fără a executa operații de introducere (PUSH) sau scoatere (POP) din stivă (tabelul 9.16*b*). Pointerul de stivă funcționează ca un numărător sincron bidirecțional cu intrări separate pentru controlul operațiilor PUSH/POP și al validării stivei, PUP respectiv $\overline{FE}$ (tabelul 9.16*b*). Dacă intrarea $\overline{FE}$ este „0" și intrarea PUP este „1" se validează o operație de introducere în stivă (PUSH). Aceasta face ca pointerul de stivă să fie incrementat și adresa de revenire (adresa următoare adresei microinstrucțiunii care a efectuat operația PUSH) să fie scrisă în stivă. Dacă intrările $\overline{FE}$ și PUP sînt „0" se validează o operație de scoatere din stivă (POP). Ieșirea stivei se poate utiliza ca adresă de revenire în ciclul de microinstrucțiune curent. Următorul front pozitiv al ceasului, începutul ciclului următor, va decrementa pointerul de stivă modificînd adresa locației punctate, deci ieșirea stivei. Dacă intrarea de validare $\overline{FE}$ este „1" pointerul de stivă este inhibat indiferent de valoarea celorlalte intrări, PUP și CP. Deoarece stiva are o capacitate de patru cuvinte este permisă înlănțuirea (*nesting*) a maximum patru subrutine.

Ieșirea Y a secvențiatorului, adresa memoriei de microprograme, depinde de semnalele OR, $\overline{ZERO}$ și $\overline{OE}$ (tabelul 9.16*c*). Intrarea $\overline{ZERO}$ se utilizează pentru a forța ieșirile Y la „0" indiferent de valoarea celorlalte intrări, cu excepția intrării $\overline{OE}$. Intrările OR servesc pentru a forța ieșirile Y la „1" permițînd implementarea anumitor instrucțiuni de salt (v. și § 10.1 și § 10.2). Ieșirile Y ale secvențiatorului sînt de tip „trei-stări" fiind validate cu ajutorul semnalului $\overline{OE}$. Acest lucru poate fi util dacă se dorește controlul extern al memoriei de microprograme. Secvențiatoarele interne se trec în starea a treia, de impedanță mare, permițîndu-se generarea externă a adresei de microprogram cu scopul, de exemplu, de a executa diverse microsubrutine de testare.

9.2.5.2. Instrucțiunile de adresare ale secvențiatoarelor Am 2909/2911 și ale circuitului Am 29811

În tabelul 9.17 se ilustrează modul în care, cu ajutorul semnalelor S_1, S_0, $\overline{FE}$ și PUP, se pot obține diverse instrucțiuni de adresare pentru secvențiatoarele Am 2909/2911. Cele patru semnale definesc adresa care se generează la ieșirile Y și starea registrelor interne după aplicarea ceasului, frontul pozitiv. În tabelul 9.17 se presupune că în ciclul N numărătorul de microprogram

Tabelul 9.17. Instrucțiuni de adresare Am 2909/2911

Ciclu	S_1	S_0	$\overline{FE}$	PUP	μPC	AR	$Stiva_0$	$Stiva_1$	$Stiva_2$	$Stiva_3$	Y	Utilizare
N	0	0	0	0	J	K	R_a	R_b	R_c	R_d	J	Extragere din stivă,
N+1		—			J+1	K	R_b	R_c	R_d	R_a	—	sfîrșit de buclă.
N	0	0	0	1	J	K	R_a	R_b	R_c	R_d	J	Introducere în stivă,
N+1		—			J+1	K	J	R_a	R_b	R_c	—	inițializare buclă
N	0	0	1	X	J	K	R_a	R_b	R_c	R_d	J	*Continue*
N+1		—			J+1	K	R_a	R_b	R_c	R_d	—	
N	0	1	0	0	J	K	R_a	R_b	R_c	R_d	K	Extragere din stivă
N+1		—			K+1	K	R_b	R_c	R_d	R_a	—	și salt pe AR, sfîr-
												șit buclă
N	0	1	0	1	J	K	R_a	R_b	R_c	R_d	K	Introducere în stivă și
N+1		—			K+1	K	J	R_a	R_b	R_c	—	salt pe AR, salt la
												subrutină (JSR AR)
N	0	1	1	X	J	K	R_a	R_b	R_c	R_d	K	Salt la adresa din AR,
N+1		—			K+1	K	R_a	R_b	R_c	R_d	—	salturi necondiționate.
N	1	0	0	0	J	K	R_a	R_b	R_c	R_d	R_a	Salt la adresa din
N+1		—			R_a+1	K	R_b	R_c	R_d	R_a	—	stivă și extragere din
												stivă, revenire din
												subrutină
N	1	0	0	1	J	K	R_a	R_b	R_c	R_d	R_a	Salt la adresa din
N+1		—			R_a+1	K	J	R_a	R_b	R_c	—	stivă și introducere în
												stivă
N	1	0	1	X	J	K	R_a	R_b	R_c	R_d	R_a	Salt la adresa din
N+1		—			R_a+1	K	R_a	R_b	R_c	R_d	—	stivă, revenire din
												buclă
N	1	1	0	0	J	K	R_a	R_b	R_c	R_d	D	Extragere din stivă
N+1		—			D+1	K	R_b	R_c	R_d	R_a	—	și salt la adresa D,
												sfîrșit de buclă
N	1	1	0	1	J	K	R_a	R_b	R_c	R_d	D	Introducere în stivă
N+1		—			D+1	K	J	R_a	R_b	R_c	—	și salt la adresa D,
												salt la subrutină
N	1	1	1	X	J	K	R_a	R_b	R_c	R_d	D	Salt la adresa D, salt
N+1		—			D+1	K	R_a	R_b	R_c	R_d	—	necondiționat.

μPC conține informația J, registrul de adresă AR, informația K și stiva informațiile R_a—R_d. După aplicarea ceasului, deci în ciclul N + 1, starea registrelor interne este definită de cele patru semnale S_1, S_0, $\overline{FE}$, PUP. De exemplu în cazul instrucțiunii JSR AR, salt la subrutina de la adresa dată de AR, în ciclul N adresa Y este egală cu conținutul registrului AR, μPC are valoarea J, iar stiva conține R_a—R_d. În ciclul N+1, μPC va fi egal cu valoarea adresei generate la ieșirile Y in crementează cu 1, deci K+1, iar stiva va conține J, R_a—R_c, deci adresa de revenire din subrutină.

Circuitul Am 29811 este destinat generării semnalelor de control și validare pentru secvențiatoare construite cu Am 2911 (figura 10.20). Funcționarea circuitului este determinată de intrările de instrucțiune I_3—I_0. Circuitul generează semnalele de control S_1, S_0, $\overline{FE}$, PUP și semnalele de validare $\overline{COUNTER\ LOAD}$, încărcare numărător, $\overline{COUNTER\ ENABLE}$, validare numărător, $\overline{MAP\ ENABLE}$, validare secvență, $\overline{PIPELINE\ ENABLE}$, validare registru *pipeline*. Instrucțiunile de adresare implementate cu Am 29811 sînt date în tabelul 9.18. Aceste instrucțiuni sînt similare cu instrucțiunile de adresare ale secvențiatorului Am 2910, descrise detaliat în paragraful următor.

Tabelul 9.18. **Instrucțiuni de adresare Am 2911 realizate cu Am 29811**

Cod mnemonic	I_3	I_2	I_1	I_0	Instrucțiunea
JZ	0	0	0	0	Salt la adresa zero
CJS	0	0	0	1	Salt condiționat la subrutină cu adresa de salt în registrul *pipeline*
JMAP	0	0	1	0	Salt la adresa de la ieșirea PROM-ului de secvențe, de *mapare*
CJP	0	0	1	1	Salt condiționat la adresa din registrul *pipeline*
PUSH	0	1	0	0	Introducere în stivă și încărcare condiționată a numărătorului
JSRP	0	1	0	1	Salt la subrutină cu adresa selectată condiționat între registrul AR din Am 2911 și registrul *pipeline*
CJV	0	1	1	0	Salt condiționat la adresa vector
JRP	0	1	1	1	Salt la adresa selectată condiționat între registrul AR din Am 2911 și registrul *pipeline*
RFCT	1	0	0	0	Repetă bucla dacă numărătorul este ≠ zero
RPCT	1	0	0	1	Repetă adresa *pipeline* dacă numărătorul ≠ zero
CRTN	1	0	1	0	Revenire condiționată din subrutină
CJPP	1	0	1	1	Salt condiționat la adresa din registrul *pipeline* și extragere din stivă
LDCT	1	1	0	0	Încărcare registru și *Continue*
LOOP	1	1	0	1	Test sfîrșit de buclă
CONT	1	1	1	0	*Continue*
JP	1	1	1	1	Salt la adresa din registrul *pipeline*

9.2.6. CONTROLORUL DE MICROPROGRAM Am 2910

Controlorul de microprogram Am 2910 generează o adresă fixă de microprogram putînd adresa pînă la 4096 de microinstrucțiuni. Deci, spre deosebire de Am 2909/2911, circuitul nu poate fi conectat în cascadă pentru obținerea unor adrese de microprogram mai mari de 12 biți.

Schema-bloc a controlorului de microprogram este dată în figura 9.10.

Am 2910 implementează 16 instrucțiuni de adresare (tabelul 9.19).

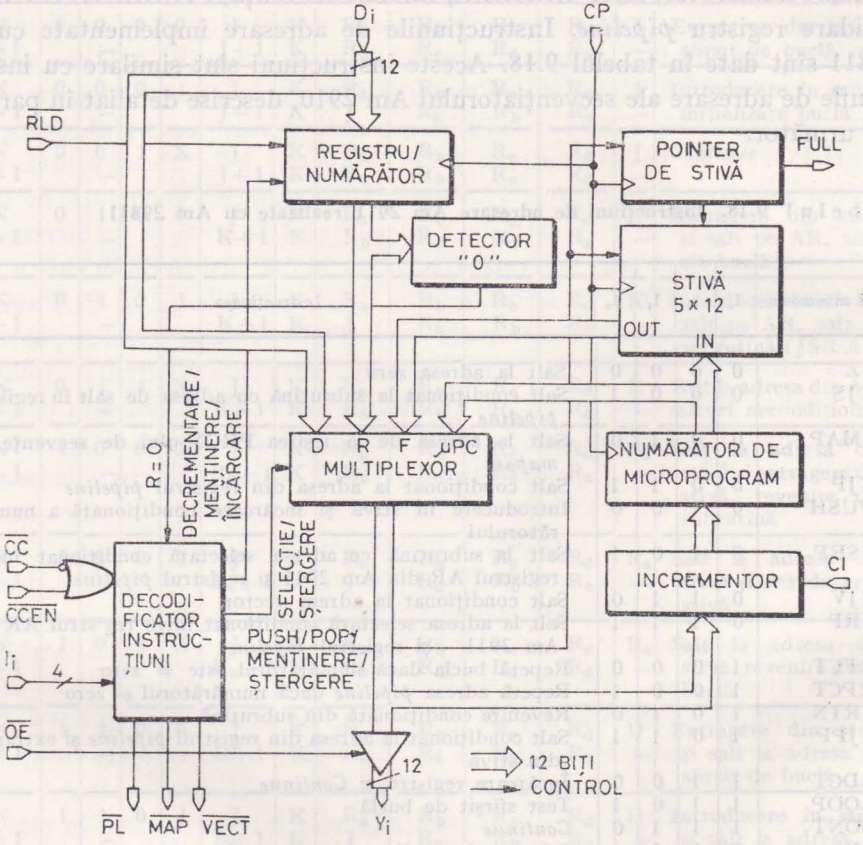


Fig. 9.10. Schema-bloc a controlorului de microprogram Am 2910

Tabelul 9.19. **Instrucțiunile de adresare ale lui Am 2910**

I_3—I_0 (zecimal)	Instrucțiunea	REG./NUM.	Test nereușit $\overline{CCEN}=0$ și $\overline{CC}=1$		Test reușit $\overline{CCEN}=1$ sau $\overline{CC}=0$		REG./NUM.	Validare
			Y	Stiva	Y	Stiva		
0	JZ	X	0	INIT.	0	INIT.	—	PL
1	CJS	X	PC	—	D	PUSH	—	PL
2	JMAP	X	D	—	D	—	—	MAP
3	CJP	X	PC	—	D	—	—	PL
4	PUSH	X	PC	PUSH	PC	PUSH	Nota 2	PL
5	JSRP	X	R	PUSH	D	PUSH	—	PL
6	CJV	X	PC	—	D	—	—	VECT
7	JRP	X	R	—	D	—	—	PL
8	RFCT	$\neq 0$	F	—	F	—	DECR.	PL
		$=0$	PC	POP	PC	POP	—	PL
9	RPCT	$\neq 0$	D	—	D	—	DECR.	PL
		$=0$	PC	—	PC	—	—	PL
10	CRTN	X	PC	—	F	POP	—	PL
11	CJPP	X	PC	—	D	POP	—	PL
12	LDCT	X	PC	—	PC	—	ÎNC.	PL
13	LOOP	X	F	—	PC	POP	—	PL
14	CONT	X	PC	—	PC	—	—	PL
15	TWB	$\neq 0$	F	—	PC	POP	DECR.	PL
		$=0$	D	POP	PC	POP	—	PL

Note: 1. „—" nu se modifică
X nu are importanță
DECR. decrementare
ÎNC. încărcare
INIT. inițializare

2. nu se modifică dacă $\overline{CCEN}=0$ și $\overline{CC}=1$, în celelalte cazuri ÎNC.

9.2.6.1. Structura circuitului

Controlorul este alcătuit dintr-un multiplexor de selectare a adresei, dintr-un registru/numărător, o stivă de 5 cuvinte, un numărător de microprogram și un PLA pentru decodificarea instrucțiunilor. Multiplexorul poate selecta ca adresă de microprogram intrarea directă D, ieșirea registrului/numărător, ieșirea stivei sau ieșirea numărătorului de microprogram.

Registrul/numărător este alcătuit din 12 bistabili de tip D acționați pe frontul pozitiv al ceasului CP. Dacă semnalul de validare a încărcării $\overline{RLD}$ este pe „0" registrul se încarcă sincron cu informația prezentă la intrările directe D, indiferent de operația specificată de instrucțiunea I_0—I_3.

Registrul/numărător poate funcționa și ca numărător-decrementor, semnalul de zero, generat de detectorul de zero asociat, servind la implementarea instrucțiunilor de buclare. Pentru a executa o secvență de microinstrucțiuni de N ori numărătorul trebuie încărcat cu valoarea N—1.

Numărătorul de microprogram este compus dintr-un registru de 12 biți și un incrementor pe 12 biți. Ca și la Am 2909/2911, incrementorul poate fi controlat cu ajutorul intrării de transport CI. Dacă CI este „1" registrul de microprogram este încărcat sincron, pe frontul pozitiv al ceasului, cu adresa curentă incrementată cu 1 (Y+1). După CI este „0" incrementorul nu mo-

difică adresa Y, astfel încît aceeași microinstrucțiune se execută pînă cînd CI devine „1".

A treia sursă pentru adresa de microprogram este ieșirea stivei utilizată pentru revenirea din microsubrutine sau pentru execuția buclelor de microprogram. Ca și la Am 2909/2911 stiva este adresată de un pointer de stivă care punctează ultima locație scrisă, ceea ce permite buclări de microprogram fără efectuarea unei operații de scoatere din stivă (instrucțiunile RFCT, RPCT). Pointerul de stivă funcționează ca un numărător bidirecțional incrementat de o operație PUSH și decrementat de o operație POP. Capacitatea stivei, 5 cuvinte, permite apelarea înlănțuită (*nesting*) a maximum 5 subrutine. După apelarea celei de-a cincea subrutine ieșirea $\overline{FULL}$ devine „0" semnalînd umplerea stivei. În acest caz orice operație de introducere în stivă (PUSH) scrie numai în vîrful stivei, lăsînd nemodificat pointerul de stivă. Această operație, în general distructivă, este de obicei interzisă. Operația de scoatere (POP) din stiva goală este, de asemenea, validă, dar trebuie ținut cont că la ieșirile Y ale controlorului se va plasa o informație fără sens, pointerul de stivă rămînînd pe zero, nemodificat.

Ultima sursă pentru adresa de microprogram este constituită de intrările directe D, utilizate de obicei ca adresă de salt.

Controlorul Am 2910 generează în exterior semnalele de validare $\overline{PL}$, $\overline{MAP}$ și $\overline{VECT}$ cu ajutorul cărora se pot selecta sursele intrării directe D (v și § 10.2). Semnalul $\overline{PL}$ este destinat pentru selecția primei surse a intrării directe D, de obicei registrul *pipeline* de microinstrucțiune. Semnalul $\overline{MAP}$ selectează a doua sursă, de regulă un PROM sau un PLA pentru generarea adreselor de început, de *mapare*, ale secvențelor de microprogram. Semnalul $\overline{VECT}$ se poate utiliza pentru selecția unei a treia surse a intrărilor D, de exemplu vectorul de întrerupere sau o adresă DMA.

Am 2910 este prevăzut cu o intrare de testare a condițiilor, $\overline{CC}$, la care se poate conecta, de exemplu, ieșirea CT a circuitului Am 2904 (v. § 9.2.4). Testul este adevărat dacă $\overline{CC}$ este „0". Intrarea $\overline{CCEN}$ invalidează intrarea $\overline{CC}$, controlorul funcționînd în starea în care testul este reușit (ca și cum $\overline{CC}$ ar fi „0").

La fel ca la Am 2909/2911, ieșirile Y de tip „trei-stări" permit controlul din exterior al structurii microprogramate a cărei funcție de secvențiere este implementată cu Am 2910. $\overline{OE}$ pe „1" invalidează ieșirile controlorului Am 2910 pe busul „trei-stări" de adresă permițînd adresarea din exterior a memoriei de microprograme.

În tabelul 9.19 sînt descrise sintetic instrucțiunile executate de Am 2910 indicîndu-se starea ieșirilor Y, a semnalelor de validare $\overline{PL}$, $\overline{MAP}$ și $\overline{VECT}$ generate în exterior, a registrului/numărător și a stivei. Valoarea efectivă a adresei de microprogram de la ieșirile Y este selectată, așa cum s-a mai spus, prin intermediul unui multiplexor. Informația încărcată în numărătorul de microprogram depinde de valoarea intrării CI, putînd fi ieșirea Y sau ieșirea Y incrementată cu 1. Orice instrucțiune validează, poziționînd pe „0", numai una din ieșirile $\overline{PL}$, $\overline{MAP}$ și $\overline{VECT}$.

9.2.6.2. Instrucțiunile lui Am 2910

Controlorul Am 2910 poate adresa microinstrucțiunea următoare cu ajutorul a 16 instrucțiuni (tabelul 9.19). Aceste instrucțiuni sînt de trei feluri: necondiționate, al căror efect asupra adresei este dat numai de instrucțiune; condiționate, al căror efect asupra adresei depinde și de o informație exterioară controlorului; instrucțiuni al căror efect este controlat de conținutul registrului/numărător. În cele ce urmează se va presupune că intrarea CI este pe „1", deci că incrementorul funcționează încărcînd Y + 1 în registrul de microprogram.

La instrucțiunile de salt condiționat rezultatul testului de care depinde adresa efectivă se aplică pe intrarea $\overline{CC}$. Dacă această intrare este „0" se consideră că testul este adevărat, reușit. În acest caz apare acțiunea specificată de numele instrucțiunii. Dacă intrarea $\overline{CC}$ este „1" testul este fals, nereușit, executîndu-se o acțiune alternativă, de obicei microinstrucțiunea următoare secvențial. Testarea internă a intrării $\overline{CC}$ poate fi invalidată prin poziționarea pe „1", prin microprogram, a semnalului $\overline{CCEN}$. În acest caz se „forțează" acțiunea specificată de numele instrucțiunii. Pentru a nu folosi un bit de microinstrucțiune pentru controlul semnalului $\overline{CCEN}$ se poate recurge la următoarele artificii:

— conectarea la „1" a intrării $\overline{CCEN}$, dacă nu există microinstrucțiuni condiționate;

— conectarea intrării $\overline{CCEN}$ la „0", dacă instrucțiunile condiționate nu sînt forțate niciodată să funcționeze necondiționat;

— conectarea intrării $\overline{CCEN}$ împreună cu bitul de instrucțiune I_0, ceea ce permite utilizarea instrucțiunilor condiționate PUSH, CJV și CRTN.

În instrucțiunile condiționate, folosite pentru bucle, care utilizează registrul/numărător, acesta este decrementat atît timp cît nu e zero. În ciclul de microinstrucțiune în care registrul/ numărător devine zero, instrucțiunea selectează o altă adresă următoare ieșind din bucla de microprogram.

Figura 9.11 ilustrează cele 16 instrucțiuni ale controlorului de microprogram Am 2910, prin indicarea succesiunii adreselor de microprogram. Valoarea adreselor alese nu are nici o importanță cu excepția instrucțiunii JZ. Exemplele se referă la o structură de tip *pipeline*, fiecare punct reprezentînd timpul cît conținutul microinstrucțiunii se află în registrul *pipeline*.

Instrucțiunea 0, JZ (*J*UMP *Z*ERO sau RESET), este o instrucțiune de salt necondiționat în care adresa microinstrucțiunii următoare este zero. Această instrucțiune se poate utiliza în secvența de punere sub tensiune pentru generarea adresei de început a microprogramului.

Instrucțiunea 1, CJS (*C*ONDITIONAL *J*UMP-TO-*S*UBROUTINE), este o instrucțiune de salt condiționat la adresa din registrul *pipeline*. Cînd conținutul microinstrucțiunii de la adresa 52 este în registrul *pipeline* se va executa o instrucțiune CJS. În acest caz adresa următoare depinde de rezultatul unui test. Dacă testul reușește, microinstrucțiunea ce va fi executată în continuare,

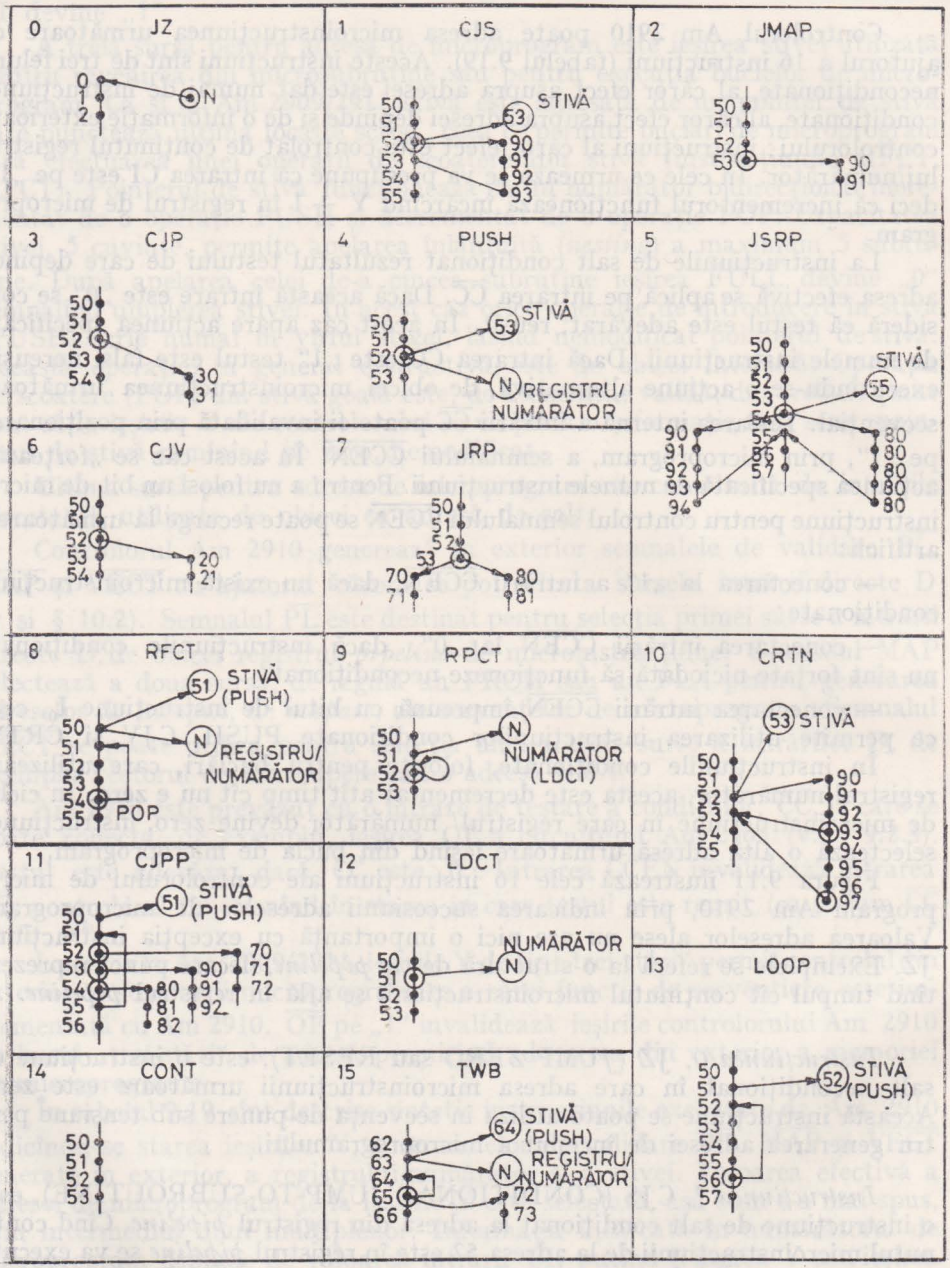


Fig. 9.11. Instrucțiunile de adresare ale controlorului Am 2910

deci care se va încărca în registrul *pipeline*, va fi cea de la adresa 90 generată de intrările directe D, unde este validat cu ajutorul semnalului $\overline{PL}$ registrul *pipeline* (cîmpul de branșare, de constantă sau de adresă următoare, al microinstrucțiunii). Adresa 53 a microinstrucțiunii următoare, adresa de revenire din subrutină, este introdusă în stivă. Revenirea din subrutină se face la adresa 93 prin execuția unei instrucțiuni RETURN-FROM-SUBROUTINE. Dacă testul nu reușește, următoarea microinstrucțiune executată va fi cea de la adresa 53.

Instrucțiunea 2, JMAP (*JUMP MAP*), este o instrucțiune de salt necondiționat. Adresa de salt este dată de intrările directe D, unde se validează cu ajutorul semnalului $\overline{MAP}$ PROM-ul sau PLA-ul pentru generarea adreselor de început ale diverselor secvențe de microprogram. Instrucțiunea JMAP se poate utiliza, de exemplu, într-o unitate centrală microprogramată la sfîrșitul secvenței de extragere (FETCH) a instrucțiunii-mașină, pentru a continua microprogramul cu secvența specifică de tratare a instrucțiunii respective. În exemplul din figura 9.11 secvența de extragere se termină cu microinstrucțiunea de la adresa 53 care conține codul instrucțiunii JMAP. Următoarea microinstrucțiune executată, cea de la adresa 90, va fi microinstrucțiunea de început a unei secvențe specifice instrucțiunii-mașină citite anterior. Adresa acestei noi microinstrucțiuni este dată la intrările directe D, de un PROM sau un PLA validate de instrucțiunea JMAP.

Instrucțiunea 3, CJP (CONDITIONAL *JUMP PIPELINE*), este o instrucțiune de salt condiționat. Această instrucțiune permite salturi în microprogram condiționate de valoarea indicatorilor de condiție reuniți la intrarea $\overline{CC}$ a controlorului. După cum se știe, mașinile microprogramate, mașinile programate în general, testează în diverse puncte de funcționare anumiți indicatori, uneori așteptînd ca aceștia să-și modifice valoarea. Atunci cînd testul reușește, indicatorul și-a modificat valoarea, microprogramul sare pentru a executa altă secvență de microinstrucțiuni specifică rezultatului testării. În figura 9.11 saltul condiționat CJP se execută de microinstrucțiunea de la adresa 52. Adresa următoare va fi dată fie de conținutul numărătorului de microprogram, fie de intrările directe D la care se validează registrul *pipeline*. Dacă testul reușește adresa următoare este cea din registrul *pipeline* (30). Dacă testul nu reușește adresa următoare este cea dată de numărătorul de program (53).

Instrucțiunea 4, PUSH (*PUSH*/CONDITIONAL LOAD COUNTER), este o instrucțiune utilizată în principal la inițializarea buclelor de microprogram. Instrucțiunea PUSH, executată în microinstrucțiunea 52, va introduce adresa următoare (53) în stivă și va încărca în mod condiționat registrul/numărător. Dacă testul nu reușește registrul/numărător nu se încarcă. Dacă testul reușește registrul/numărător se va încărca cu valoarea dată la intrările directe D de registrul *pipeline* validat de semnalul $\overline{PL}$. Instrucțiunea RFCT poate utiliza rezultatele instrucțiunii PUSH (adresa introdusă în stivă

și numărul de bucle introdus în registrul/numărător) pentru execuția efectivă a buclelor de microprogram.

Instrucțiunea 5, JSRP (CONDITIONAL JUMP-TO-SUBROUTINE REGISTER/PIPELINE), este o instrucțiune de salt condiționat. Instrucțiunea, așa cum se vede și în figura 9.11, introduce întotdeauna adresa următoare (55) în stivă și execută condiționat una din cele două subrutine (cea de la adresa 80 sau cea de la adresa 90). Pentru ca instrucțiunea JSRP să fie corect executată trebuie avut grijă ca registrul/numărător să fie anterior încărcat cu valoarea dorită. În figura 9.11 registrul/numărător poate fi încărcat, de exemplu, cu valoarea 90, de microinstrucțiunea de la adresa 53. La execuția microinstrucțiunii de la adresa 54 adresa subrutinei ce va fi executată în continuare depinde de rezultatul testului de condiție. Dacă testul reușește adresa următoare este dată de cîmpul de branșare din registrul *pipeline* (80). Dacă testul nu reușește adresa următoare este dată de conținutul registrului/numărător (încărcat anterior cu valoarea 90).

Instrucțiunea 6, CJV (CONDITIONAL JUMP VECTOR), este o instrucțiune de salt condiționat la o adresă validată la intrările directe D cu ajutorul semnalului de validare $\overline{\text{VECT}}$. Această instrucțiune permite generarea adreselor subrutinelor de tratare a întreruperilor. Pentru că instrucțiunea este condiționată, adresa următoare depinde de rezultatul testului făcut asupra intrării $\overline{\text{CC}}$. Dacă intrarea $\overline{\text{CC}}$ este „0", testul reușit, adresa următoare este generată din exterior la intrările directe D (20). Dacă intrarea $\overline{\text{CC}}$ este „1", testul nereușit, adresa următoare (53) este dată de numărătorul de microprogram.

Instrucțiunea 7, JRP (CONDITIONAL JUMP REGISTER/PIPELINE), este o instrucțiune de salt condiționat similară cu instrucțiunea JSRP. Diferența constă în faptul că instrucțiunea JRP nu execută nici o operație asupra stivei. În exemplul din figura 9.11 se presupune că după încărcarea microinstrucțiunii de la adresa 53 în registrul *pipeline*, registrul/numărător este deja încărcat cu valoarea utilă 70. Cîmpul de salt al microinstrucțiunii 53 constituie cealaltă adresă utilă (80) execuției instrucțiunii JRP. În acest fel, în funcție de rezultatul testului, microinstrucțiunea executată în continuare va fi cea de la adresa 70, testul nereușit, sau cea de la adresa 80, testul reușit.

Instrucțiunea 8, RFCT (REPEAT LOOP, FILE, COUNTER≠ZERO), este o instrucțiune de salt condiționat utilizată în implementarea buclelor de microprogram. Pentru a fi corect executată trebuie ca în prealabil registrul/numărător și stiva să fie încărcate cu numărul de buclări, respectiv cu adresa de început a buclei. Aceasta se poate face, de exemplu, cu o instrucțiune PUSH. Instrucțiunea RFCT verifică dacă registrul/numărător este diferit de zero. Dacă registrul/numărător este diferit de zero, acesta se decrementează, iar adresa microinstrucțiunii următoare este luată din vîrful stivei (adresa de început a buclei). Operația de citire din stivă nu este însoțită de o decre-

mentare a pointerului de stivă, ceea ce permite reluarea buclei pînă cînd registrul/numărător devine zero. Atunci cînd se ajunge în această situaţie, condiţia de ieşire din buclă, adresa microinstrucţiunii următoare este dată de numărătorul de microprogram. Stiva se modifică, se reface, prin execuţia unei operaţii POP. În exemplul din figura 9.11 microinstrucţiunea de la adresa 50 execută o instrucţiune PUSH introducînd valoarea N în registrul/numărător (numărul de bucle = N + 1) şi adresa 51, adresa de început a buclei, în stivă. Deoarece testarea numărului de bucle se face la sfîrşitul buclei, în microinstrucţiunea 54, valoarea care trebuie încărcată în registrul/numărător este numărul dorit de buclări plus 1. În acest fel se pot executa cu ajutorul controlorului Am 2910 bucle de 1 ÷ 4096 ori. Dacă se doreşte să se execute bucle de 0 ÷ 4095 ori, testarea ieşirii din buclă trebuie făcută la începutul buclei.

Instrucţiunea 9, RPCT (*R*EPEAT LOOP, *P*IPELINE REGISTER, COUN*T*ER≠ZERO), este o instrucţiune similară cu instrucţiunea RFCT. Diferenţa constă în faptul că adresa de început a buclei este dată de intrările directe D, unde este validat cîmpul de salt din microinstrucţiunea aflată în registrul *pipeline*. Uneori această instrucţiune este folosită ca un mijloc de extensie a stivei deoarece ea permite execuţia unei bucle, chiar după ce stiva este plină. La sfîrşitul buclei instrucţiunea RPCT nu modifică stiva. În exemplul din figura 9.11 bucla conţine o singură microinstrucţiune (52) care va executa şi instrucţiunea de adresare RPCT. Cîmpul de salt al microinstrucţiunii trebuie să aibă valoarea 52, adresa de început a buclei. Ca şi în cazul instrucţiunii RFCT registrul/numărător trebuie încărcat în prealabil cu numărul dorit de buclări. Acest lucru este făcut de microinstrucţiunea 51 cu ajutorul unei instrucţiuni LDCT.

Instrucţiunea 10, CRTN (CONDITIONAL *RETURN*), este o instrucţiune de salt condiţionat utilizată pentru ieşire din subrutine şi revenire la microinstrucţiunea următoare microinstrucţiunii apelante. Revenirea se face numai dacă testul reuşeşte. Dacă testul nu reuşeşte se execută microinstrucţiunea următoare secvenţial. Deşi este o instrucţiune condiţionată, prin microprogram se poate forţa o funcţionare necondiţionată, poziţionînd semnalul $\overline{CCEN}$ pe „1". Apelarea subrutinei în microinstrucţiunea 52 presupune introducerea în stivă a adresei 53 şi continuarea microprogramului cu prima microinstrucţiune a subrutinei apelate (90). Microinstrucţiunea 93 execută o CRTN condiţionată. Dacă testul reuşeşte se efectuează o operaţie POP şi adresa microinstrucţiunii următoare va fi 53. Dacă testul nu reuşeşte se va executa microinstrucţiunea 94. Microprogramul va continua pînă la sfîrşitul subrutinei unde va întîlni din nou o instrucţiune CRTN. Pentru ca instrucţiunea să fie executată necondiţionat trebuie avut grijă ca microinstrucţiunea respectivă să poziţioneze intrarea de validare $\overline{CCEN}$ pe „1".

Instrucțiunea 11, CJPP (CONDITIONAL JUMP PIPELINE AND POP), este o instrucțiune de salt condiționat. Instrucțiunea poate fi utilizată la ieșirea din buclele de microprogram sau în diverse operații cu stiva. În figura 9.11 bucla de microprogram este formată din microinstrucțiunile 51—55. Microinstrucțiunile de la adresele 52, 53, 54 sînt toate de tip CJPP. În acest fel se pot asigura mai multe puncte de ieșire condiționată dintr-o buclă de microprogram. De exemplu, la adresa 52 dacă intrarea $\overline{CC}$ este pe „0", testul reușit, se va face un salt la adresa 70. Pointerul de stivă va fi decrementat păstrîndu-se astfel corectitudinea stivei (anterior, de exemplu la adresa 50, se introdusese în stivă adresa de început a buclei). Dacă testul nu reușește, $\overline{CC} = 1$, se va executa microinstrucțiunea 53. În același fel se poate ieși din buclă la adresele 53 și 54. Instrucțiunea CJPP este foarte utilă atunci cînd bucla de microprogram testează mai mulți indicatori așteptînd modificarea unuia dintre ei. Instrucțiunea CJPP permite, de asemenea, utilizarea comodă în implementarea microprogramelor a cunoscutei tehnici de programare cu tabele de salturi.

Instrucțiunea 12, LDCT (LOAD COUNTER AND CONTINUE), este o instrucțiune necondiționată care permite încărcarea registrului/numărător cu informația prezentă la intrările directe D. Intrările D sînt de obicei conectate la cîmpul de constantă al microinstrucțiunii aflată în registrul *pipeline*. Acest cîmp de constantă poate servi ca adresă de salt sau pentru încărcarea registrului/numărător. După cum s-a văzut există trei moduri de a încărca registrul/numărător:

— o încărcare simplă cu ajutorul instrucțiunii LDCT;

— o încărcare condiționată cu instrucțiunea PUSH;

— o încărcare „hardware" cu ajutorul semnalului $\overline{RLD}$ indiferent de instrucțiunea $I_0 — I_3$.

Instrucțiunea LDCT este de fapt o combinație între instrucțiunea CONT și $\overline{RLD}=0$ cu scopul de a asigura o posibilitate simplă de încărcare a registrului/numărător, fără a mai fi nevoie de controlul prin microprogram al intrării $\overline{RLD}$.

Instrucțiunea 13, LOOP (TEST END-OF-*LOOP*), este o instrucțiune de salt condiționat. LOOP permite ieșirea condiționată dintr-o buclă, la sfîrșitul ei. Dacă testul este reușit se reia bucla de la adresa dată de ieșirea stivei. Dacă testul nu este reușit microprogramul se continuă secvențial. În exemplul din figura 9.11 instrucțiunea LOOP se execută la adresa 56. Dacă testul nu reușește microprogramul va sări la adresa 52 încărcată anterior în stivă, în microinstrucțiunea de la adresa 51, printr-o instrucțiune PUSH. Dacă testul reușește microprogramul iese din buclă trecînd la microinstrucțiunea 57 și refăcînd stiva prin decrementarea pointerului de stivă.

Instrucțiunea 14, CONT (*CONT*INUE), permite execuția secvențială a microprogramului. Această instrucțiune de adresare este cea mai simplă, ea putînd fi considerată în unele situații și ca o instrucțiune neoperantă.

Instrucțiunea 15, TWB (*T*HREE-*W*AY *B*RANCH), constituie o instrucțiune de salt condiționat. Este cea mai complexă instrucțiune de adresare a controlorului Am 2910 asigurînd atît testarea intrării $\overline{CC}$ cît și a registrului/numărător. Instrucțiunea selectează ca adresă următoare între ieșirea stivei, registrul *pipeline* și numărătorul de microprogram. Ca și în cazul instrucțiunii RFCT trebuie încărcate în prealabil numărul de buclări în registrul/numărător și adresa de început a buclei în stivă. TWB efectuează ca și RFCT operații de decrementare și salt pînă cînd registrul/numărător devine zero. Adresa de salt este luată din vîrful stivei. După ce registrul/numărător devine zero, adresa următoare este dată de registrul *pipeline*. Acest mod de execuție corespunde situației în care testul nu reușește. Dacă testul reușește nu se mai face nici un salt și adresa următoare este dată de numărătorul de microprogram. La ieșirea din buclă, pe registrul/numărător zero sau pe test reușit, stiva se reface prin decrementarea pointerului de stivă. În figura 9.11 se dă un exemplu de utilizare a instrucțiunii TWB în cazul unei operații de căutare într-un tabel. Microinstrucțiunea 63 execută o operație PUSH introducînd adresa 64 în stivă și numărul N în registrul/numărător (lungimea tabelei — 1). Microinstrucțiunea de la adresa 64 extrage din tabel elementul următor și îl compară cu cheia de căutare. Microinstrucțiunea 65 testează rezultatul comparației executînd o instrucțiune de adresare TWB. Dacă elementul extras din tabel nu este cel căutat testul nu reușește și microprogramul revine la adresa 64 pentru o nouă citire în tabel. Dacă registrul/numărător devine zero se sare la adresa 72 unde începe secvența specifică situației în care elementul căutat nu a fost găsit în tabel. Dacă elementul căutat se găsește în tabel microprogramul iese din bucla de căutare executînd microinstrucțiunea 66 care reprezintă începutul secvenței specifice acestei situații. La ieșirea din buclă instrucțiunea TWB reface stiva decrementînd pointerul de stivă.

9.2.7. CONTROLORUL DE PROGRAM Am 2930

Controlorul de program Am 2930 este un circuit expandabil destinat implementării funcției de adresare la nivel de program (adresarea instrucțiunilor) sau la nivel de microprogram (adresarea microinstrucțiunilor).

9.2.7.1. Structura circuitului

Schema-bloc a circuitului se dă în figura 9.12.

Circuitul este compus din următoarele elemente principale:

— numărător de program (PC);
— stivă LIFO pentru subrutine (S);
— registru de lucru (R);
— sumator;
— decodificator de instrucțiuni.

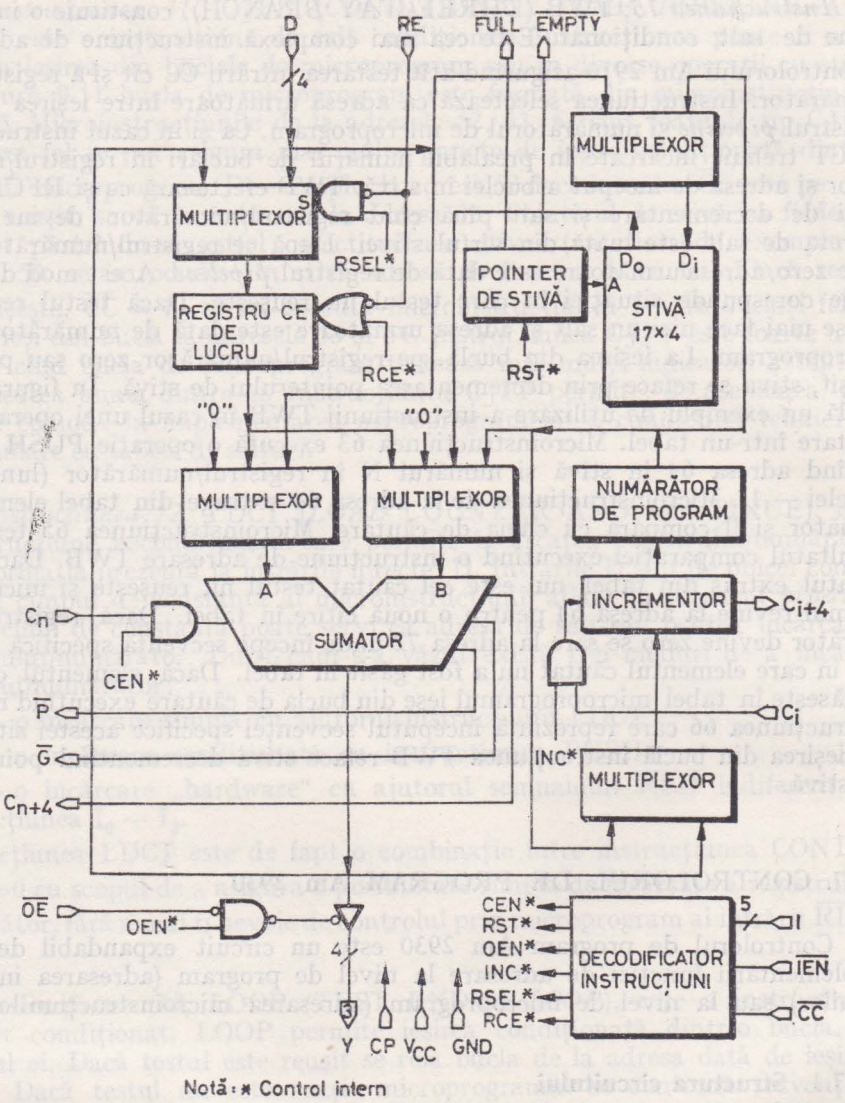


Fig. 9.12. Schema-bloc a controlorului de program Am 2930

Numărătorul de program este alcătuit dintr-un registru, un incrementor și un multiplexor de intrare. Registrul de patru bistabili de tip D este acționat pe frontul pozitiv al ceasului CP. Incrementorul este realizat cu transport anticipat pentru mărirea vitezei circuitului. La conectarea în cascadă ieșirea C_{i+4} a incrementorului se conectează la intrarea de transport C_i a circuitului următor. Ieșirea incrementorului este egală cu intrarea plus C_i. Este deci posibil să se controleze întregul incrementor prin intermediul intrării C_i a

celui mai puțin semnificativ circuit. Dacă această intrare este pe „0" incrementorul va lăsa nemodificată ieșirea multiplexorului, iar dacă este pe „1" o va incrementa cu 1. Incrementorul este inhibat intern, în timpul anumitor instrucțiuni de adresare, de semnalul INC. Multiplexorul permite selectarea informației de intrare în incrementor între PC și sumator.

Stiva LIFO este alcătuită dintr-un multiplexor de intrare, o memorie RAM de 17 × 4 biți și un pointer (SP) pentru adresare. SP punctează întotdeauna ultimul cuvînt scris în RAM, vîrful stivei, disponibil deci la ieșirea S a stivei. Informația care se scrie în stivă poate fi selectată cu ajutorul multiplexorului între intrările directe D și PC. Scrierea în stivă se face la adresa SP + 1, iar citirea, la adresa SP. Pointerul SP este un numărător bidirecțional acționat pe frontul pozitiv al ceasului. Atunci cînd stiva este plină se inhibă incrementarea SP, iar cînd stiva este goală, decrementarea lui. De asemenea, atunci cînd stiva este plină se inhibă și operația de scriere în RAM. Cînd stiva este goală ieșirea $\overline{\text{EMPTY}}$ este poziționată pe „0", iar atunci cînd este plină, ieșirea $\overline{\text{FULL}}$ se poziționează pe „0".

Registrul de lucru poate fi încărcat fie cu informația prezentă la intrările directe D, fie cu ieșirea sumatorului. Este acționat pe frontul pozitiv al ceasului și este validat de intrarea externă $\overline{\text{RE}}$ sau, intern, de semnalul RSEL, în timpul instrucțiunilor de adresare. Intrarea $\overline{\text{RE}}$ este prioritară validînd, atunci cînd este „0", încărcarea în R a intrărilor D, indiferent de instrucțiune.

Sumatorul binar pe patru biți are, ca și numărătorul de program, transport anticipat pentru mărirea vitezei circuitului. În vederea obținerii transportului anticipat într-o conectare în cascadă, Am 2930 generează semnalele $\overline{\text{P}}$ și $\overline{\text{G}}$. Pentru sisteme mai lente se poate conecta ieșirea C_{n+4} la intrarea C_n a următorului circuit obținîndu-se, în acest fel, o structură cu transport-serie. Multiplexoarele de la intrările sumatorului permit selectarea perechilor de operanzi în funcție de instrucțiunea de adresare.

Decodificatorul de instrucțiuni generează semnalele necesare stabilirii căilor de informație și validării operațiilor de încărcare în PC, R, SP, RAM. Intrarea de condiție $\overline{\text{CC}}$ este utilizată pentru decodificare numai în instrucțiuni de salt condiționat ($\overline{\text{CC}}$=0 reprezintă testul reușit). Intrarea $\overline{\text{IEN}}$ de validare-instrucțiune inhibă operațiile cu PC, SP și de scriere în stivă, atunci cînd este pe „1". De asemenea, atunci cînd $\overline{\text{IEN}}$ = 1 registrul R este controlat numai de intrarea de validare $\overline{\text{RE}}$, indiferent de instrucțiunea de adresare.

9.2.7.2. Instrucțiunile de adresare ale controlorului Am 2930

Am 2930 poate executa 32 de instrucțiuni de adresare specificate de biții de instrucțiune I_0—I_4 (tabelul 9.20). Instrucțiunile de adresare ale Am 2930 pot fi împărțite în:

— instrucțiuni de extragere necondiționată;

Tabelul 9.20. **Instrucțiunile de adresare ale controlorului Am 2930**

Cod mnemonic	I_4	I_3	I_2	I_1	I_0	$\overline{CC}$	$\overline{IEN}$	Instrucțiunea	Y_{0-3}	Starea următoare				
										PC	R		RAM	SP
											$\overline{RE}=0$	$\overline{RE}=1$		
	X	X	X	X	X	X	1	Invalidare	Nota 2	—	D	—	—	—
PRST	0	0	0	0	0	X	0	RESET	„0"	„0" + C_i	D	—	—	Reset
FPC	0	0	0	0	1	X	0	FETCH PC	PC	PC + C_i	D	—	—	—
FR	0	0	0	1	0	X	0	FETCH R	R	PC + C_i	D	—	—	—
FD	0	0	0	1	1	X	0	FETCH D	D	PC + C_i	D	—	—	—
FRD	0	0	1	0	0	X	0	FETCH R + D	R + D + C_n	PC + C_i	D	—	—	—
FPD	0	0	1	0	1	X	0	FETCH PC + D	PC + D + C_n	PC + C_i	D	—	—	—
FPR	0	0	1	1	0	X	0	FETCH PC + R	PC + R + C_n	PC + C_i	D	—	—	—
FSD	0	0	1	1	1	X	0	FETCH S + D	S + D + C_n	PC + C_i	D	—	—	—
FPLR	0	1	0	0	0	X	0	FETCH PC→R	PC	PC + C_i	PC	PC	—	—
FRDR	0	1	0	0	1	X	0	FETCH R+D→R	R + D + C_n	PC + C_i	R + D + C_n	R + D + C_n	—	—
PLDR	0	1	0	1	0	X	0	LOAD R	PC	PC + C_i	D	D	—	—
PSHP	0	1	0	1	1	X	0	PUSH PC	PC	PC + C_i	D	—	PC→(SP + 1)	SP + 1
PSHD	0	1	1	0	0	X	0	PUSH D	PC	PC + C_i	D	—	D→(SP + 1)	SP + 1
POPS	0	1	1	0	1	X	0	POP S	S	PC + C_i	D	—	—	SP − 1
POPP	0	1	1	1	0	X	0	POP PC	PC	PC + C_i	D	—	—	SP − 1
PHLD	0	1	1	1	1	X	0	HOLD	PC	—	D	—	—	—

Tabelul 9.20 *(continuare)*

	1	X	X	X	X	1	0	Test nereuşit	PC	PC+C_i	D	–	–	–
JMPR	1	0	0	0	0	0	0	JUMP R	R	R+C_i	D	–	–	–
JMPD	1	0	0	0	1	0	0	JUMP D	D	D+C_i	D	–	–	–
JMPZ	1	0	0	1	0	0	0	JUMP „0“	„0“	„0“+C_i	D	–	–	–
JPRD	1	0	0	1	1	0	0	JUMP R+D	R+D+C_n	R+D+C_n+C_i	D	–	–	–
JPPD	1	0	1	0	0	0	0	JUMP PC+D	PC+D+C_n	PC+D+C_n+C_i	D	–	–	–
JPPR	1	0	1	0	1	0	0	JUMP PC+R	PC+R+C_n	PC+R+C_n+C_i	D	–	–	–
JSBR	1	0	1	1	0	0	0	JSB R	R	R+C_i	D	–	PC→(SP+1)	SP+1
JSBD	1	0	1	1	1	0	0	JSB D	D	D+C_i	D	–	PC→(SP+1)	SP+1
JSBZ	1	1	0	0	0	0	0	JSB „0“	„0“	„0“+C_i	D	–	PC→(SP+1)	SP+1
JSRD	1	1	0	0	1	0	0	JSB R+D	R+D+C_n	R+D+C_n+C_i	D	–	PC→(SP+1)	SP+1
JSPD	1	1	0	1	0	0	0	JSB PC+D	PC+D+C_n	PC+D+C_n+C_i	D	–	PC→(SP+1)	SP+1
JSPR	1	1	0	1	1	0	0	JSB PC+R	PC+R+C_n	PC+R+C_n+C_i	D	–	PC→(SP+1)	SP+1
RTS	1	1	1	0	0	0	0	RETURN S	S	S+C_i	D	–	–	SP−1
RTSD	1	1	1	0	1	0	0	RETURN S+D	S+D+C_n	S+D+C_n+C_i	D	–	–	SP−1
CHLD	1	1	1	1	0	0	0	HOLD	PC	–	D	–	–	–
PSUS	1	1	1	1	1	0	0	SUSPEND	Hi-Z	–	D	–	–	–

Nota 1: PC – numărător de program; SP – pointerul de stivă;
R – registrul auxiliar; D – intrările directe; Hi-Z – starea a treia; – = nici o modificare

Nota 2: Ieşirile *Y* sînt determinate ca şi în cazul $\overline{IEN}=0$ de I_0-I_4 şi $\overline{CC}$

— instrucțiuni de salt condiționat;
— instrucțiuni de salt condiționat la subrutină;
— instrucțiuni de revenire condiționată din subrutină;
— instrucțiuni diverse.

Instrucțiunile de extragere necondiționată (1 ÷ 9) permit citirea (extragerea) la ieșirile Y ale circuitului a unor diverse elemente din interiorul lui (PC, R, D, S). În tabelul 9.20 se observă că pentru toate instrucțiunile de extragere PC este incrementat cu valoarea intrării C_i a celui mai puțin semnificativ circuit (într-o structură cu mai multe Am 2930 conectate în cascadă). Pentru instrucțiunile 1÷7 registrul de lucru R este controlat de intrarea de validare $\overline{RE}$. Pentru instrucțiunile 8 și 9 registrul R este încărcat cu PC, respectiv R + D. În timpul instrucțiunilor de extragere stiva nu este modificată.

Instrucțiunile de salt condiționat (16÷21) permit selectarea adresei următoare în funcție de intrarea de test $\overline{CC}$. Dacă testul reușește, $\overline{CC} = 0$, adresa următoare este o funcție de elementele din interiorul controlorului (R, D, PC, „0"). Se observă în tabelul 9.20 că această funcție, selectată prin instrucțiunea de adresare, eventual incrementată cu 1, este încărcată în PC. Dacă testul nu reșește, $\overline{CC} = 1$, se va executa o operație FETCH PC, adresa următoare fiind dată de PC. Instrucțiunile de salt condiționat nu modifică stiva.

Instrucțiunile de salt condiționat la subrutină (22÷27) permit apelarea condiționată a subrutinelor. Dacă testul nu reușește, $\overline{CC} = 1$, adresa următoare este dată de PC; stiva nu se modifică. Dacă testul reușește, $\overline{CC} = 0$, la ieșirile Y ale circuitului se plasează o funcție de elementele din interiorul lui (R, D, PC, „0"). Ieșirile Y, eventual incrementate (în funcție de C_i), sînt încărcate în PC, PC fiind încărcat în stivă la locația SP + 1 — operație prin care se incrementează și pointerul. În timpul acestor instrucțiuni registrul R este controlat prin intermediul intrării de validare $\overline{RE}$.

Instrucțiunile de revenire condiționată din subrutină (28, 29) permit ieșirea condiționată din subrutină. Dacă testul nu reușește, $\overline{CC} = 1$, adresa următoare este dată de PC, stiva rămînînd nemodificată. Dacă testul reușește, $\overline{CC} = 0$, adresa următoare este S, respectiv S + D. Această adresă, eventual incrementată, în funcție de C_i, este încărcată în PC, iar pointerul de stivă decrementat. Registrul R este controlat în timpul acestor instrucțiuni prin intermediul intrării de validare $\overline{RE}$.

Instrucțiunile diverse (0, 10—15, 30, 31) cuprind:
— instrucțiunea PRST (RESET) care forțează ieșirile Y ale circuitului pe zero, încarcă C_i în PC și șterge SP;
— instrucțiunea PLDR (LOAD R) care încarcă intrările directe D în registrul R și, eventual, incrementează PC, în funcție de C_i;

— instrucțiunea PSHP (PUSH PC) care, ca și FETCH PC (tabelul 9.20), plasează la ieșirile Y numărătorul de program și, eventual, incrementează PC; PSHP introduce PC în locația SP+1 din stivă și incrementează pointerul;

— instrucțiunea PSHD (PUSH D), asemănătoare cu PUSH PC, plasează la ieșirile Y numărătorul de program și, eventual, îl incrementează; deosebirea constă în faptul că în stivă se introduce informația prezentă la intrările directe D;

— instrucțiunea POPS (POP S) plasează la ieșirile Y informația din vîrful stivei, decrementează pointerul și, eventual, incrementează PC;

— instrucțiunea POPP (POP PC) plasează la ieșirile Y numărătorul de program, decrementează pointerul de stivă și, eventual, incrementează PC;

— instrucțiunea PHLD (HOLD) plasează la ieșirile Y numărătorul de program PC lăsînd nemodificate numărătorul de program și stiva;

— instrucțiunea CHLD (CONDITIONAL HOLD) care, atunci cînd testul este reușit, $\overline{CC} = 0$, acționează ca PHLD, iar cînd testul nu este reușit, $\overline{CC} = 1$, ca FPC;

— instrucțiune PSUS (SUSPEND) care forțează condiționat ieșirile Y în starea Hi-Z de impedanță mare, starea a treia.

9.3. ALTE FAMILII ȘI CIRCUITE *BIT-SLICE*

9.3.1. SERIA Intel 3000

9.3.1.1. Controlorul de microprogram Intel 3001

Intel 3001 este un circuit destinat implementării funcției de secvențiere într-o structură de control microprogramată [2,3].

Schema-bloc a circuitului este dată în figura 9.13.

Controlorul îndeplinește în principal două tipuri de funcțiuni:

— funcțiuni de control al adresei microinstrucțiunii următoare;

— funcțiuni de control al indicatorilor de condiție.

Corespunzător acestor funcțiuni Intel 3001 conține un registru-adresă-microprogram împreună cu logica de selecție a adresei următoare, respectiv bistabilii C-flag, Z-flag și *latch*-urile F.

Logica de selecție a adresei următoare implementează un set de instrucțiuni de adresare condiționate și necondiționate. Cu scopul minimizării numărului de ieșiri ale controlorului și al reducerii logicii de selecție a adresei următoare, Intel a proiectat circuitul 3001 pentru a adresa memorii de microprograme organizate matricial. Fiecare adresă de microprogram este împărțită într-o adresă de rînd și o adresă de coloană. Un controlor Intel 3001 poate adresa, cu ajutorul celor 9 biți de adresă MA_8—MA_0, maximum 512 microinstrucțiuni organizate într-o matrice de 32 rînduri și 16 coloane. Logica de selecție a adresei următoare este simplificată datorită folosirii acestei scheme

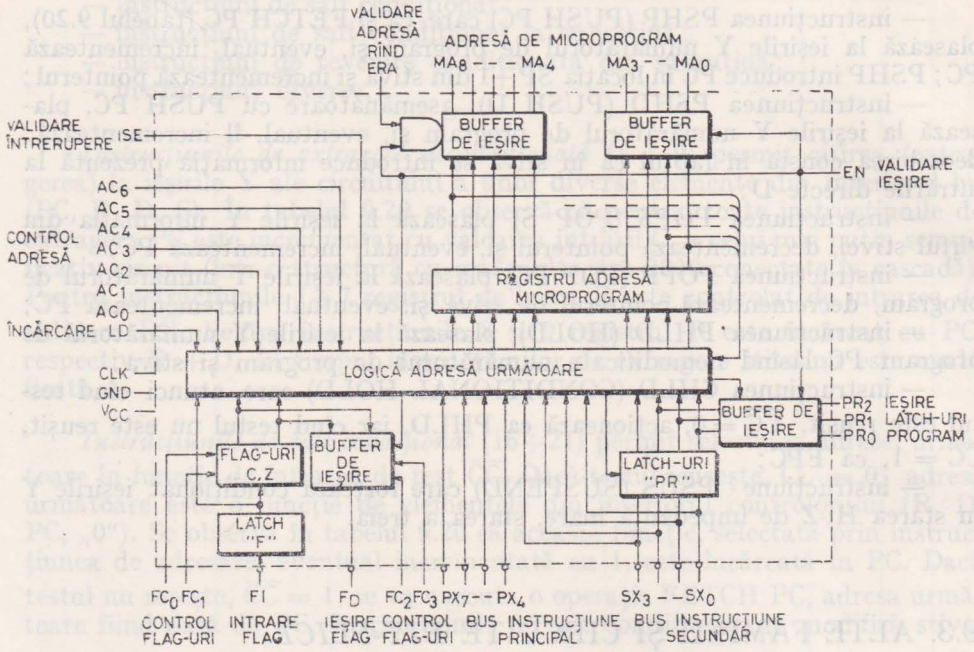


Fig. 9.13. Schema-bloc a controlorului de microprogram Intel 3001.

de adresare matricială. De exemplu, pentru o anumită adresă de microinstrucțiune, microprogramul poate sări necondiționat oriunde pe linia sau coloana respectivă, dar nu este posibil să sară oriunde în întreg spațiul de memorie. Cu alte cuvinte microprogramul nu poate sări decît într-o anumită „zonă de salt" *(jump set)*. Fiecare instrucțiune de adresare are o zonă de salt asociată.

Logica de condiție, bistabilii C-flag, Z-flag și *latch*-urile F, permit implementarea unui set de funcțiuni care asigură salvarea valorii curente a ieșirii de transport a procesorului și controlul intrării de transport în procesor. Aceste două tipuri de funcțiuni distincte sînt numite funcțiuni de intrare-condiții și funcțiuni de ieșire-condiții. Logica de condiție se poate utiliza împreună cu logica pentru deplasări și transport din procesor cu scopul implementării unor diverse funcțiuni aritmetice și/sau de deplasare/rotire.

Instrucțiunile de adresare sînt selectate cu ajutorul intrărilor AC_0—AC_6. Adresa generată de logica de selecție a adresei următoare este încărcată sincron pe frontul pozitiv al ceasului în registrul-adresă-microprogram. Adresa efectivă spre memoria de microprograme, MA_0—MA_8, este generată prin intermediul unor *driver*-e. Așa cum s-a mai spus, adresa microinstrucțiunii următoare este împărțită în două părți: adresa de rînd, MA_8—MA_4, și adresa de coloană, MA_3—MA_0.

Controlul indicatorilor de condiție se face cu ajutorul intrărilor FC_0—FC_3.

Încărcarea registrului adresă microprogram cu informația de pe *bus*-urile de instrucțiune PX_4—PX_7 și SX_0—SX_3 se face pe frontul pozitiv al ceasului

CLK atunci cînd intrarea de validare LD este pe „1". PX_4—PX_7 se încarcă în MA_0—MA_3, SX_0—SX_3 în MA_4—MA_7, iar MA_8 se poziționează pe „0". Deci, se vede că informația de pe *bus*-ul principal de instrucțiune PX_4—PX_7 poate specifica adresa coloanei, iar informația de pe *bus*-ul secundar SX_0—SX_3, adresa rîndului.

Validarea întreruperilor se face prin poziționarea ieșirii ISE (*I*NTERRUPT *S*TROBE *E*NABLE). Ieșirea ISE este poziționată pe „1" de instrucțiunea de adresare JZR în coloana 15. Microinstrucțiunea de pe rîndul zero/coloana 15 poate reprezenta, de exemplu, începutul unui ciclu EXTRAGERE/EXECUȚIE atunci cînd controlorul 3001 este utilizat pentru implementarea microprogramată a unei unități centrale. Deci JZR în coloana 15, prin poziționarea ieșirii ISE, poate valida un circuit de control al întreruperilor, de exemplu Intel 3214. Circuitul de control al întreruperilor răspunde la o întrerupere prin poziționarea semnalului ERA (*E*NABLE *R*OW *A*DDRESS) pe zero, ceea ce va invalida *driver*-ele de ieșire pentru adresa de rînd. În acest fel se permite generarea din exteriorul circuitului 3001 a adreselor de rînd unde încep subrutinele de tratare a întreruperilor.

Funcția de încărcare, cu ajutorul semnalului LD, este întotdeauna prioritară față de controlul adresei prin intermediul intrărilor AC_0—AC_6. Totuși această încărcare nu este mai prioritară decît citirea sau încărcarea *latch*-urilor PR cu ajutorul instrucțiunilor JCE, respectiv JPX. De asemenea, încărcarea registrului-adresă-microprogram nu perturbă controlul întreruperilor, poziționarea ISE și controlul indicatorilor de condiție.

9.3.1.2. Microprocesorul Intel 3002

Intel 3002 este un microprocesor *bit-slice* de 2 biți conectabil în cascadă. Schema-bloc a circuitului este dată în figura 9.14.

După cum se vede, Intel 3002 este format din următoarele elemente principale:

— decodificatorul de microfuncțiuni;
— unitatea aritmetică-logică, UAL, și multiplexoarele A, B;
— registrele de lucru R_0—R_3 și T;
— registrul acumulator AC;
— registrul-adresă-memorie MAR.

Decodificatorul de microfuncțiuni decodifică intrările F_0—F_6 pentru a genera semnalele de selecție a operației UAL, adresa registrului de lucru și semnalele de control pentru multiplexoarele A și B.

Unitatea aritmetică-logică efectuează operații aritmetice și logice. Rezultatul unei operații UAL poate fi încărcat în registrul acumulator sau într-unul din registrele de lucru. Intrarea LI (LEFT IN) și ieșirea RO (RIGHT OUT) sînt utilizate în operațiile de deplasare dreapta. Intrarea și ieșirea de transport, CI, respectiv CO, servesc pentru cuplarea circuitului într-o structură lentă cu transport-serie. Pentru o structură rapidă cu transport anticipat

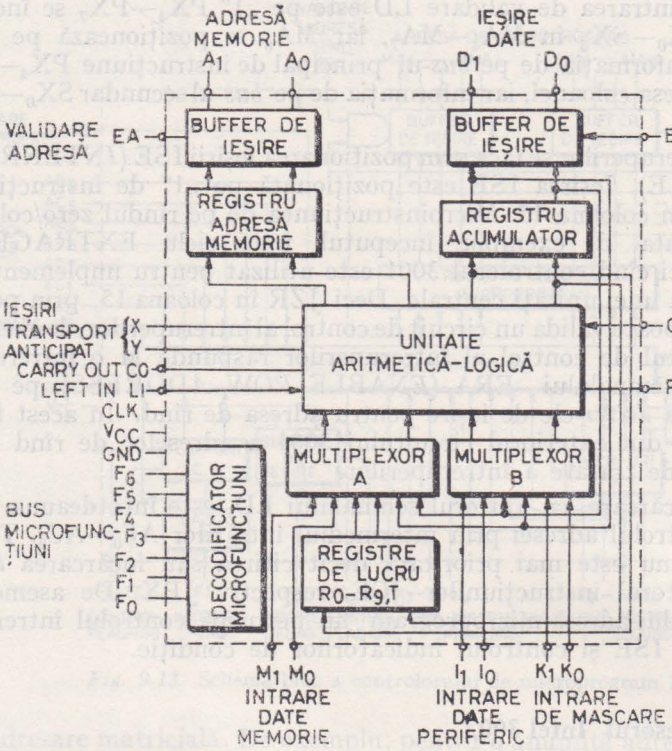


Fig. 9.14. Schema-bloc a microprocesorului *bit-slice* Intel 3002

circuitul generează semnale standard X și Y. Aceste semnale pot fi utilizate de generatorul de transport anticipat Intel 3003. Multiplexoarele A și B selectează cele două intrări în UAL. Multiplexorul A are ca intrări *bus*-ul M, registrele de lucru și acumulatorul, iar multiplexorul B, *bus*-ul I, *bus*-ul K și acumulatorul. *Bus*-ul K maschează întotdeauna celelalte intrări selectate de multiplexorul B, asigurîndu-se în acest fel un mijloc comod de mascare și testare la nivel de bit.

Registrele de lucru R_0—R_9 și T sînt încărcate cu ieșirea UAL. Ieșirile registrelor de lucru sînt multiplexate pentru a putea fi selectate ca intrare în UAL.

Registrul acumulator se încarcă cu rezultatul unei operații UAL. Acumulatorul poate fi selectat ca operand UAL sau citit pe *bus*-ul de date D. De obicei *bus*-ul D este utilizat pentru transmiterea informației spre memoria principală sau spre dispozitivele de I/E.

Registrul-adresă-memorie se încarcă cu rezultatul unei operații UAL și poate fi citit pe *bus*-ul de adresă A. MAR și *bus*-ul A pot fi utilizate pentru adresarea memoriei principale sau pentru selectarea unui dispozitiv de I/E.

9.3.2. FAMILIA F100K

Această familie de circuite LSI, realizate în tehnologie ECL de firma Fairchild, cuprinde circuite de tip *bit-slice* pe 8 biți [9].

Circuitul principal al familiei este unitatea de interfațare-date și adrese F100 220. De tip *bit-slice*, circuitul poate executa 27 de instrucțiuni principale. Familia mai cuprinde un circuit multifuncțional, F100 221, care asigură adaptarea circuitului F100 220 la restul structurii de control construite cu alte circuite, o stivă cu două *port*-uri de acces, F100 222, și o unitate de interfață programabilă, F100 223, destinată cuplării de dispozitive de I/E. Circuitul F100 223 este realizat în interior în tehnologie ECL, fiind în exterior compatibil TTL.

Familia F100 K este, pînă în prezent, singura familie de circuite *bit-slice* care lucrează pe „felii" de 8 biți. Fiind realizată în tehnologie ECL, ea este destinată implementării de structuri de control microprogramate de foarte mare viteză. De exemplu, cu ajutorul circuitelor din această familie se pot implementa unități aritmetice pe 64 biți cu transport anticipat care să execute o operație de adunare în 35 ns.

9.3.3. FAMILIA MACROLOGIC

Circuitele care intră în componența familiei MACROLOGIC [5] sînt produse de firma Fairchild în două variante:

— seria 4700, realizată în tehnologie CMOS, caracterizată, în principal, prin imunitate la zgomot mare și putere consumată foarte mică;

— seria 9400, realizată în tehnologii Schottky avansată și I^3L, caracterizată, în principal, prin viteză de lucru mai mare și compatibilitate TTL.

Familia MACROLOGIC este alcătuită din circuite de tip *bit-slice* conectabile în cascadă, destinate realizării de structuri de control microprogramabile caracterizate fie prin putere consumată mică și imunitate la zgomot mare, fie prin viteză de lucru mare. Familia este alcătuită din următoarele circuite mai importante:

— memorie FIFO* 9403 (4703);
— microprocesor *bit-slice* 9405 (4705);
— stivă de program 9406 (4706);
— secvențiator de microprogram 9408 (4708).

Microprocesorul *bit-slice* pe patru biți 9405 (4705) conține o unitate aritmetică-logică, o memorie RAM de 8 × 4 biți, un registru de ieșire, logica de control asociată. UAL implementează 8 funcții aritmetice-logice asupra a doi operanzi selectați unul din exteriorul circuitului, celălalt din memoria internă. Rezultatul operației UAL este încărcat în registrul RAM selectat ca operand și în registrul de ieșire. Circuitul poate fi conectat în cascadă într-o structură cu transport-serie sau anticipat și generează ca semnale de ieșire semnale de stare de tip semn, depășire, zero.

Secvențiatorul de microprogram 9408 (4708) generează o adresă de microprogram de zece biți (circuitul nu este de tip *bit-slice*) și poate executa 16

* First In—First Out

instrucțiuni de adresare condiționate și necondiționate. Circuitul are 7 intrări de test din care patru sînt necesare în instrucțiunile de salt condiționat, iar celelalte trei în salturile multidirecționale *(multi-way)*. Intrările pentru salturi condiționate sînt memorate în bistabili și pot fi testate individual prin instrucțiunile de adresare. Celelalte trei intrări de test formează cei mai puțin semnificativi biți ai adresei asigurîndu-se în acest fel o posibilitate de salt multidirecțional în funcție de valoarea condițiilor testate.

Memoria FIFO 9403 (4703) servește în principal ca memorie tampon în cuploarele de discuri și benzi magnetice sau în cuploarele de comunicații. Este organizată în 16 cuvinte de 4 biți și poate fi extinsă pe verticală și orizontală. Informația se poate introduce în/sau extrage din această memorie asincron și sincron, serie sau paralel.

Stiva de program 9406 (4706) este destinată implementării numărătorului de program și stivei pentru subrutine. Circutul execută patru instrucțiuni, *Return*, *Branch*, *Call*, *Fetch*, și poate fi conectat în cascadă.

9.3.4. FAMILIA M10800

Circuitele familiei Motorola M10800 sînt realizate în tehnologie ECL fiind destinate implementării de structuri de control microprogramate foarte rapide [4]. Corespunzător funcțiunilor unor astfel de structuri, familia este compusă din următoarele circuite mai importante:

— microprocesor *bit-slice* MC10800;
— secvențiator de microprogram MC10801;
— circuit pentru implementarea funcției de *timing* MC10802;
— circuit de interfață cu memoria MC10803.

MC10800 este un microprocesor pe 4 biți de tip *bit-slice* (fig. 9.15). Circuitul este alcătuit în principal dintr-o UAL, circuite de memorare și mascare, circuite pentru deplasări și un acumulator. Microprocesorul este controlat prin intermediul a 17 linii de selecție realizînd diverse funcțiuni aritmetice și logice necesare în SCM.

MC10801 este un seventiator de microprogram pe 4 biți. Circuitul de tip *bit-slice* generează adresa de microinstrucțiune, fiind alcătuit în principal dintr-un registru-adresă-memorie, registre de lucru, un incrementor, o stivă LIFO pentru subrutine, logica de selecție a adresei următoare. Funcționarea circuitului poate fi controlată cu ajutorul unui cîmp de microinstrucțiune de 4 biți și a nouă linii de selecție.

MC10802 implementează funcția de *timing* dintr-o structură de control microprogramată, generînd fazele ceasului și secvențele de punere sub tensiune, oprire sau diagnoză. Circuitul, de tip *bit-slice* pe 4 biți, poate fi conectat în cascadă pentru obținerea de ceasuri cu mai mult de patru faze. Circuitul este controlat prin intermediul a 15 linii.

MC10803 este destinat interfațării unei SCM construite cu circuite din familia M10800 cu unitatea de memorie sau cu dispozitive periferice. Circuitul, de tip *bit-slice* pe 4 biți, este alcătuit în principal din 6 registre, logica de generare a adresei de memorie, o unitate aritmetică-logică, circuite de control intern.

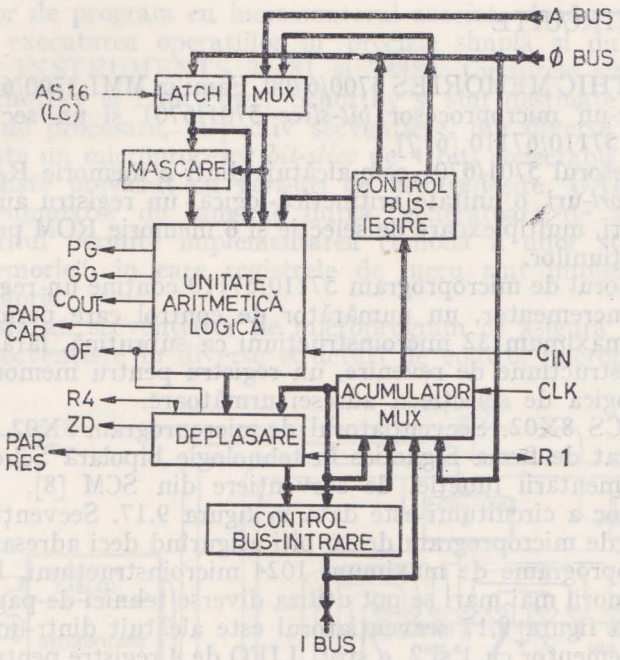


Fig. 9.15. Schema-bloc a microprocesorului *bit-slice* MC10800

Familia M10800 se caracterizează în special prin viteza mare de funcționare a circuitelor (20 MHz). Structurile realizate cu circuite din această familie, cum este cea din figura 9.16, permit obținerea unor cicli de microinstrucțiune sub 100 ns.

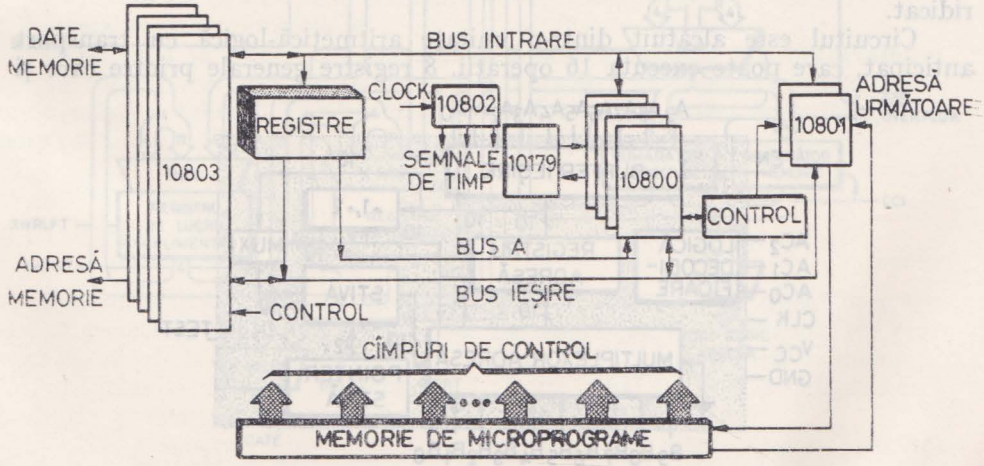


Fig. 9.16. O structură microprogramată realizată cu circuite din familia M10800.

9.3.5. ALTE CIRCUITE

MONOLITHIC MEMORIES 5700/6700. Familia MMI 5700/6700 constă în principal dintr-un microprocesor *bit-slice* 5701/6701 și un secvențiator de microprogram 57110/67110 [6, 7].

Microprocesorul 5701/6701 este alcătuit dintr-o memorie RAM de 16 ×4 biți cu două *port*-uri, o unitate aritmetică-logică, un registru auxiliar, logica pentru deplasări, multiplexoare de selecție și o memorie ROM pentru decodificarea instrucțiunilor.

Secvențiatorul de microprogram 57110/67110 conține un registru-adresă-memorie, un incrementor, un numărător de control care permite execuția unui grup de maximum 32 microinstrucțiuni ca subrutină, fără să fie necesară o microinstrucțiune de revenire, un registru pentru memorarea adresei de revenire, logica de selecție a adresei următoare.

SIGNETICS 8X02. Secvențiatorul de microprogram 8X02 este un circuit LSI realizat de firma Signetics în tehnologie bipolară Schottky și este destinat implementării funcției de secvențiere din SCM [8].

Schema-bloc a circuitului este dată în figura 9.17. Secvențiatorul generează o adresă de microprogram de 10 biți asigurînd deci adresarea unei memorii de microprograme de maximum 1024 microinstrucțiuni. Pentru adresarea unor memorii mai mari se pot utiliza diverse tehnici de paginare. După cum se vede în figura 9.17 secvențiatorul este alcătuit dintr-un registru de adresă, un incrementor cu 1 și 2, o stivă LIFO de 4 registre pentru subrutine, un multiplexor pentru selecția adresei următoare. Pentru realizarea unor SCM mai complexe secvențiatorul 8X02 se poate cupla ușor cu toate microprocesoarele *bit-slice* compatibile TTL prezentate în paragrafele precedente, 2901, 3002, 9405, 6701, precum și cu alte unități aritmetice-logice.

TEXAS INSTRUMENTS SBP 0400. SBP 0400 este un microprocesor *bit-slice* pe 4 biți realizat în tehnologie I^2L, prezentînd deci avantajele acestei tehnologii: putere consumată foarte mică la un grad de integrare relativ ridicat.

Circuitul este alcătuit dintr-o unitate aritmetică-logică cu transport anticipat, care poate executa 16 operații, 8 registre generale printre care și

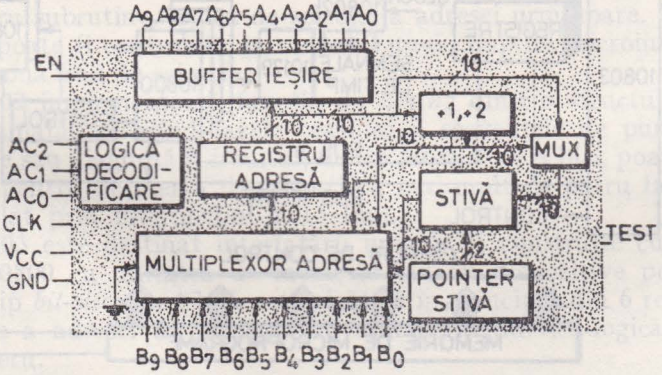


Fig. 9.17. Schema-bloc a secvențiatorului de microprogram 8X02.

un numărător de program cu incrementorul asociat, două registre auxiliare care permit executarea operațiilor în precizie simplă și dublă.

TEXAS INSTRUMENTS 74481 și 74482. Circuitele sînt realizate în tehnologii Schottky și Low Power Schottky și sînt destinate implementării funcțiunilor de procesare, respectiv secvențiere, din diverse SCM.

74481 este un microprocesor *bit-slice* pe 4 biți conectabil în cascadă (figura 9.18). Este prevăzut cu *port*-uri de intrare/ieșire, UAL cu transport anticipat, acumulator de lungime dublă, generatoare de adresă-memorie [6, 10]. Circuitul permite implementarea comodă a unor structuri de tip „memorie-memorie", în care registrele de lucru sînt implementate într-o zonă de memorie.

74482 este un secvențiator de microprogram ce asigură implementarea funcției de secvențiere în diverse structuri de control microprogramate.

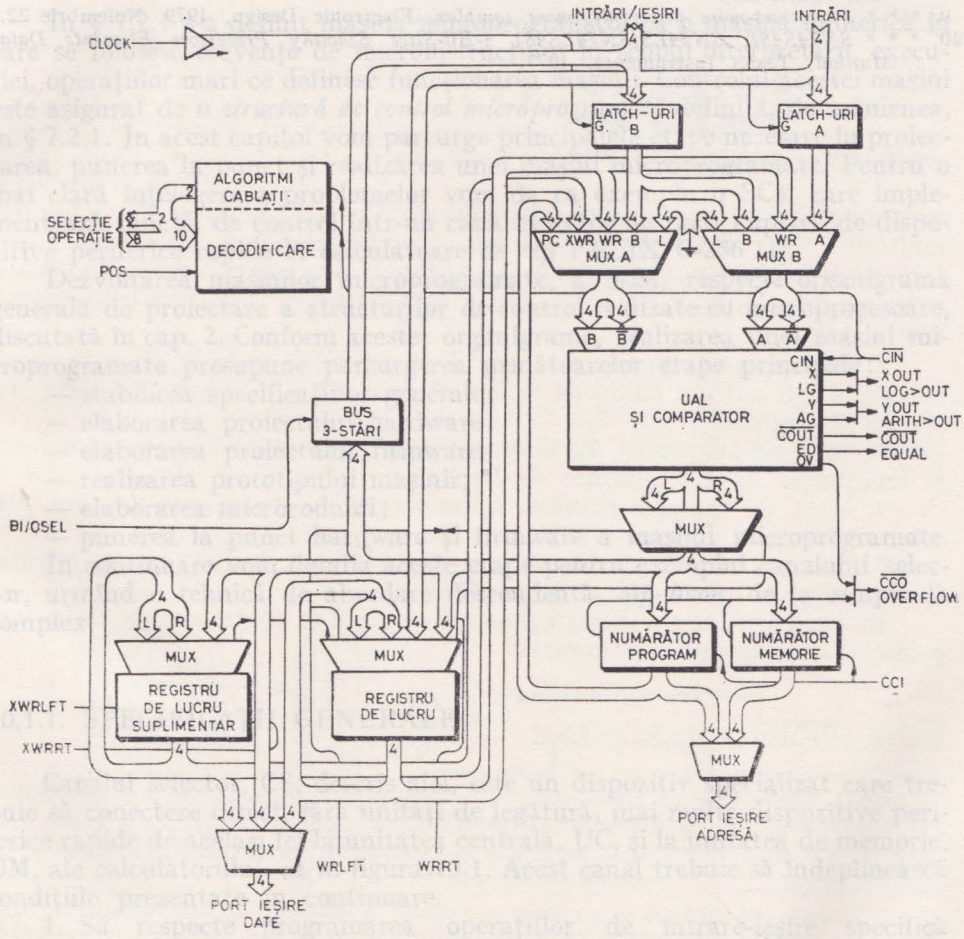


Fig. 9.18. Schema-bloc a microprocesorului *bit-slice* 74481

BIBLIOGRAFIE

1. * * * *The Am 2900 Family Data Book with Related Support Circuits*, Advanced Micro Devices Inc., Sunnyvale, California, 1979.
2. * * * *Series 3000 Reference Manual*, Santa Clara, California, Intel Corp., 1976.
3. * * * *Introducing the Series 3000 Bipolar Microprocessor* Signetics Corp., Sunnyvale, California.
4. * * * *M10800-High Performance MECL LSI Processor Family*, Motorola Semiconductor Products Inc., 1977
5. * * * *MACROLOGIC Microprocessor Databook*, Fairchild Corp., Mountain View, California, 1976.
6. RAUSHER, T.G.; ADAMS, P.M., *Microprogramming: A Tutorial and Survey of Recent Developments*, IEEE Transactions on Computers, 1980, C-29, 1, p. 2—20.
7. EDWARDS, M.; DAGLESS, E., *LSI microprogrammable microprocessors*, Microprocessors, 1977, 1, 10.
8. * * * *Signetics 8X02 Control Store Sequencer. Applications Manual*, Signetics Corp., Sunnyvale, California.
9. * * * *Processors-more powerfull, more complex*, Electronic Design, 1979, Noiembrie 22.
10. * * * *SN74S481, SN54LS/SN74LS481 4-Bit-Slice Schottky Processors Elements Data Manual*, Texas Instruments, 1977.

APLICAŢII ALE MICROPROCESOARELOR *BIT-SLICE*

10.1. PROIECTAREA UNEI MAŞINI MICROPROGRAMATE

În § 7.2.1 s-a definit maşina microprogramată ca o maşină numerică în care se folosesc secvenţe de microinstrucţiuni în scopul interpretării, execuţiei, operaţiilor mari ce definesc funcţionarea maşinii. Controlul acestei maşini este asigurat de o *structură de control microprogramată* definită, de asemenea, în § 7.2.1. În acest capitol vom parcurge principalele etape necesare în proiectarea, punerea la punct şi realizarea unei maşini microprogramate. Pentru o mai clară înţelegere a problemelor vom da ca exemplu o SCM care implementează funcţia de control într-un canal selector destinat cuplării de dispozitive periferice rapide la calculatoare de tip FELIX C-256 [1].

Dezvoltarea maşinilor microprogramate, a SCM, respectă organigrama generală de proiectare a structurilor de control realizate cu microprocesoare, discutată în cap. 2. Conform acestei organigrame, realizarea unei maşini microprogramate presupune parcurgerea următoarelor etape principale:

— stabilirea specificaţiilor generale;
— elaborarea proiectului hardware;
— elaborarea proiectului firmware;
— realizarea prototipului maşinii;
— elaborarea microcodului;
— punerea la punct hardware şi firmware a maşinii microprogramate.

În continuare vom detalia aceste etape pentru exemplul canalului selector, urmînd o tehnică de abordare descendentă, *top-down*, de la simplu la complex.

10.1.1. SPECIFICAŢII GENERALE

Canalul selector, CS, descris aici, este un dispozitiv specializat care trebuie să conecteze direct, fără unităţi de legătură, mai multe dispozitive periferice rapide de acelaşi fel la unitatea centrală, UC, şi la unitatea de memorie, UM, ale calculatorului, ca în figura 10.1. Acest canal trebuie să îndeplinească condiţiile prezentate în continuare.

1. Să respecte programarea operaţiilor de intrare-ieşire specifică sistemelor FELIX C-256. CS va fi lansat printr-o instrucţiune SIO

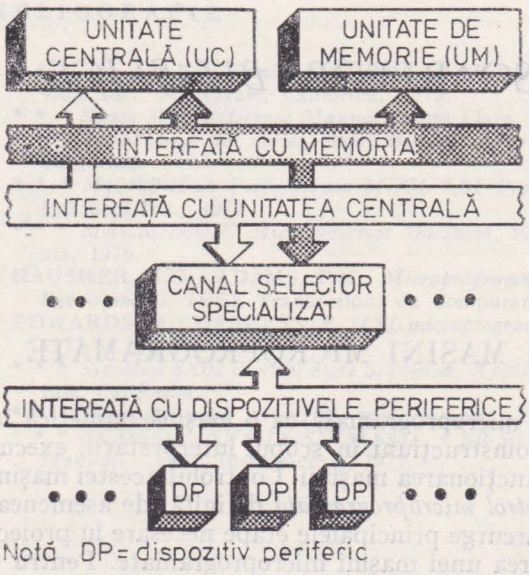


Fig. 10.1. Conectarea unui canal selector specializat într-un sistem de calcul

programabilă ca și în cazul canalelor de tip „unitate de schimburi multiple" [3]. Testarea dispozitivelor periferice și a transferului se va face prin instrucțiunile TDV și TIO, iar oprirea unei operații de intrare-ieșire, cu instrucțiunea HIO. Cererile de întrerupere se vor achita prin instrucțiunea AIO.

2. Să respecte interfețele standard cu unitatea centrală și cu memoria calculatorului atît din punct de vedere electric, cît și în ceea ce privește protocolul de transfer.

3. Întrucît adaptează direct perifericul, acest canal trebuie să realizeze și funcțiunile de tip „unitate de legătură".

4. Rata de transfer maximă a perifericelor care se pot conecta prin intermediul acestui canal este de 5 MHz.

5. Structura canalului este necesar să fie cît mai flexibilă pentru a putea fi ușor adaptat la alte tipuri de periferice.

Pe baza acestor specificații generale se trece în continuare la elaborarea proiectelor hardware și firmware.

10.1.2. PROIECT HARDWARE

În această fază a realizării unei mașini microprogramate proiectantul general trebuie să definească intrarea/ieșirea sau interfețele*, schema-bloc care să specifice structura generală a resurselor hardware, microoperațiile

* Termenul de interfață este utilizat aici pentru a desemna un set de semnale electrice și un protocol de transfer cu ajutorul cărora se controlează schimbul de informație între diferitele unități ale unui sistem.

primitive din mașina microprogramată, formatul microinstrucțiunii. Acestea vor constitui mai departe punctele de referință utilizate pentru dezvoltarea hardware-ului și firmware-ului.

Detalierile care urmează au scopul de a pregăti ca exemplu implementarea prin microcod în CS a răspunsului la instrucțiunea TIO.

10.1.2.1. Definirea interfețelor

Canalul selector respectă, conform specificațiilor generale, interfețele standard cu unitatea centrală și memoria calculatorului. Interfața cu dispozitivele periferice este particulară și nu va fi descrisă aici din motive de claritate, pentru o mai bună înțelegere a problemelor.

În figura 10.2*a* se dă interfața la nivel de cablu cu unitatea centrală. Semnificația semnalelor, negate pe cablu datorită circuitelor de interfață, este următoarea:

— CI0, 1, 2, coduri-instrucțiune, codifică instrucțiunile de I/E, SIO, TIO, HIO, TDV și AIO;

— TUC, martor UC, eșantionează codurile-instrucțiune;

— ATUC, achitare TUC, semnalul de răspuns al canalului la o instrucțiune emisă de UC;

— CC1, 2, coduri-condiție, indicatorii de condiție Z și S din instrucțiunile de tip FELIX poziționați ca răspuns la o instrucțiune de I/E;

— AUE, adresa-unitate de schimburi, indică adresa canalului;

— RZES, ștergere generală I/E;

— DI, cerere de întrerupere.

Figura 10.2*b* reprezintă o cronogramă a dialogului între UC și CS.

În figura 10.3*a* se dă interfața la nivel de cablu cu unitatea de memorie. Semnificația semnalelor, de asemenea negate pe cablu datorită circuitelor de interfață, este următoarea:

— DAM, cerere de acces la memorie;

— AD00-16, adresa-memorie;

— INF00-15, informația transferată;

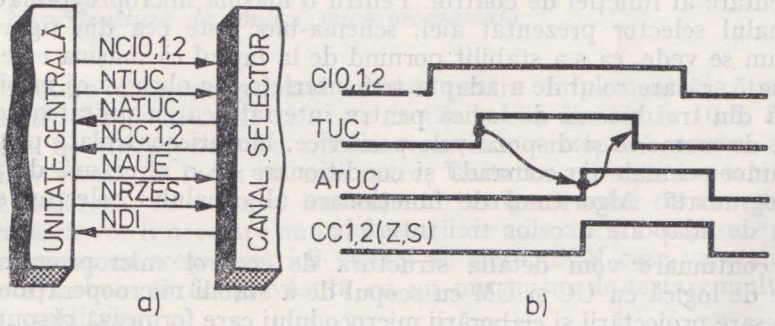


Fig. 10.2. Interfața cu unitatea centrală

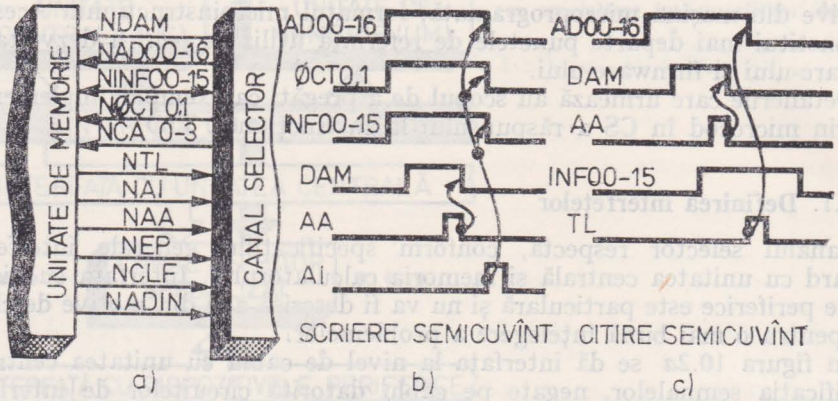


Fig. 10.3. Interfața cu unitatea de memorie

— OCT0, 1, comenzi de scriere/citire a octetului în cadrul semicuvîntului adresat de AD00-16 (00—citire semicuvînt, 01—scriere octet dreapta, 10—scriere octet stînga, 11—scriere semicuvînt);
— CA0-3, chei de acces;
— TL, martor informație citită;
— AI, achitare informație scrisă;
— AA, achitare adresă;
— EP, eroare paritate;
— CLF, chei de acces false;
— ADIN, adresă inexistentă.

Dialogul între UM și CS pentru operații de scriere, respectiv citire, este dat în figurile 10.3*b* și 10.3*c*.

10.1.2.2. Stabilirea schemei-bloc

Schema-bloc a unei mașini numerice poate fi concepută pornind de la algoritmul de funcționare global, relațiile mașinii cu exteriorul, modul de implementare al funcției de control. Pentru o mașină microprogramată, cum este canalul selector prezentat aici, schema-bloc este cea din figura 10.4. După cum se vede, ea s-a stabilit pornind de la faptul că mașina este microprogramată, că are rolul de a adapta trei interfețe. Se observă că mașina este alcătuită din trei blocuri de logică pentru interfațarea cu unitatea centrală, unitatea de memorie și dispozitivele periferice, blocuri controlate prin intermediul unor semnale de comandă și condiționare de o structură de control microprogramată. Algoritmul de funcționare al canalului selector este un algoritm de adaptare a celor trei interfețe.

În continuare vom detalia structura de control microprogramată și blocurile de logică cu UC și UM cu scopul de a stabili microoperațiile primitive necesare proiectării și elaborării microcodului care formează răspunsul CS la instrucțiunea TIO.

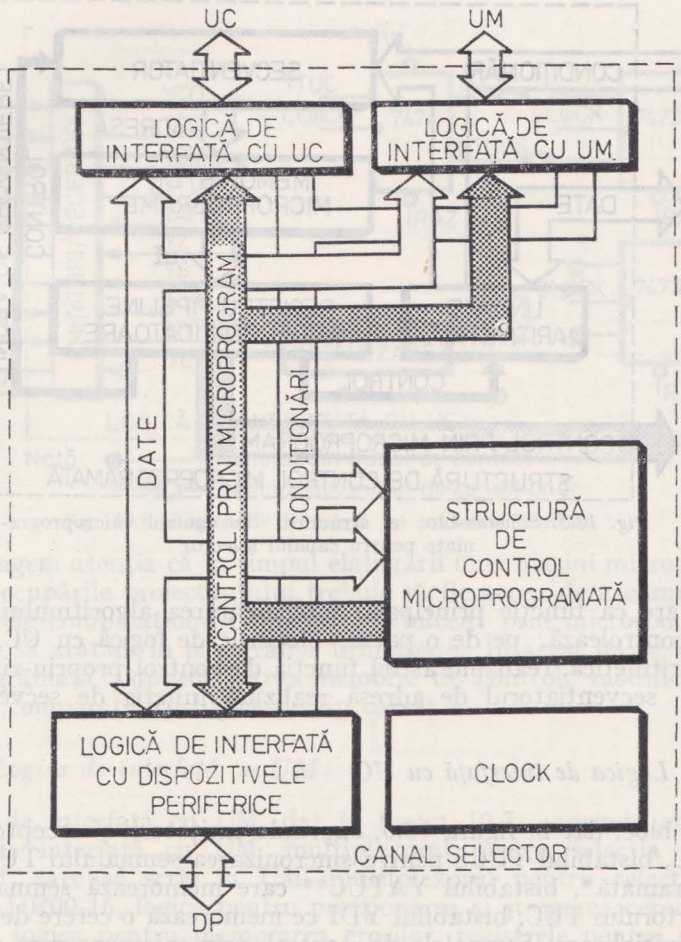


Fig. 10.4. Schema-bloc a canalului selector

10.1.2.2.1. *Structura de control microprogramată*

Structura microprogramată aleasă pentru implementarea controlului în canalul selector discutat aici este de tipul celei din figura 10.5. S-a ales o structură de tip *pipeline* cu microinstrucțiune orizontală pentru a asigura mașinii o viteză de lucru cît mai mare. Unitatea aritmetică poate fi foarte simplă deoarece, în vederea implementării algoritmului de funcționare, se estimează că vor fi necesare numai operații de adunare și comparare pe octet. De asemenea, cu scopul măririi vitezei de lucru, SCM care va realiza multe teste, va trebui să fie prevăzută cu un mecanism de testare multidirecțională a condițiilor.

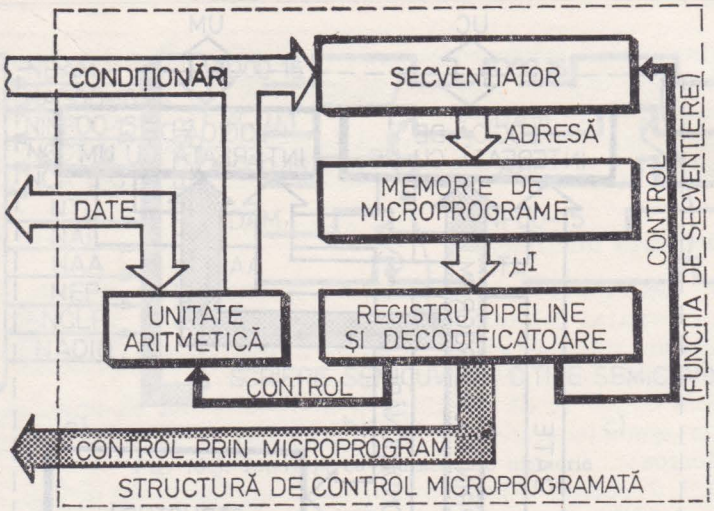


Fig. 10.5. Schema-bloc a structurii de control microprogramate pentru canalul selector

SCM are ca funcție principală implementarea algoritmului de funcționare. Ea controlează, pe de o parte, blocurile de logică cu UC, UM, DP și unitatea aritmetică, realizînd astfel funcția de control propriu-zis, iar, pe de altă parte, secvențiatorul de adresă realizînd funcția de secvențiere.

10.1.2.2.2. *Logica de interfață cu UC*

Acest bloc, dat în figura 10.6, cuprinde emițătorii și receptorii de interfață cu UC, bistabilul TUCS pentru sincronizarea semnalului TUC cu mașina microprogramată*, bistabilul YATUC** care memorează semnalul de achitare a martorului TUC, bistabilul YDI ce memorează o cerere de întrerupere, logica de inițializare. Semnalele de control prin microprogram sînt intrările de poziționare pe „1" ale bistabililor YATUC și YDI, răspunsurile YZ și YS, semnalele de inițializare. Aceste semnale de control vor fi precizate după stabilirea formatului microinstrucțiunii, pe parcursul etapei de proiectare și elaborare a microcodului. După cum se vede, logica din figura 10.6 este proiectată convențional cu ajutorul unor circuite integrate SSI și MSI. Considerăm că în acest fel se pun mai bine în evidență relațiile firmware-hardware și microoperațiile primitive ale mașinii, exemplul dat fiind prezentat mai didactic. Logica din figura 10.6 conține elemente de sincronizare și memorare controlate prin microprogram, emițători/receptori, diverse circuite combina-

* Mașina microprogramată discutată aici este o mașină sincronă. De aceea toate semnalele care intră în mașină trebuie să fie sincronizate cu ceasul intern CLOCK pentru a se asigura relații de timp certe.

** Semnalele care încep cu Y sînt semnale TTL din CS înainte/după emițătorii/receptorii de interfață cu UC.

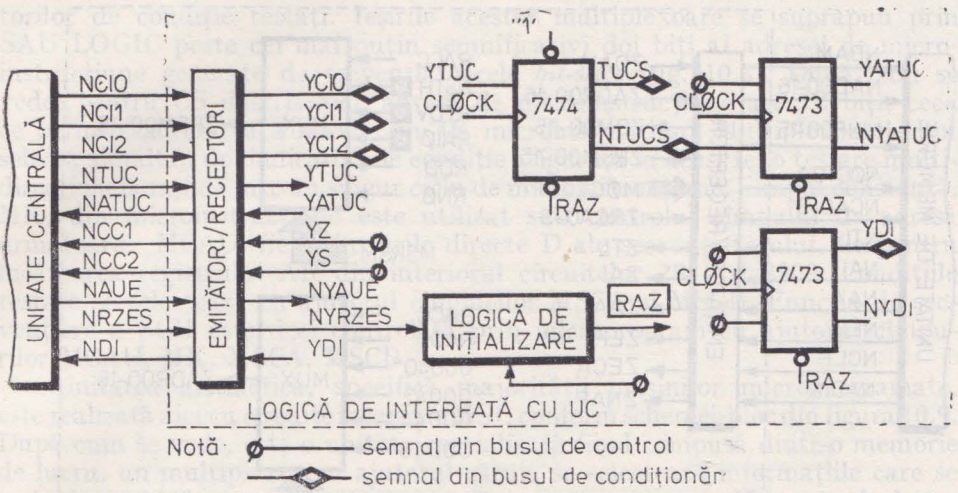


Fig. 10.6. Logica de interfață cu unitatea centrală

ționale. Atragem atenția că în timpul elaborării unei mașini microprogramate una din preocupările proiectantului trebuie să fie accea de a elimina cît mai mult circuitele combinaționale care se vor înlocui prin microcod, așa cum vom încerca să arătăm în paragrafele următoare. În acest fel se va obține o „logică programată" stocată într-o memorie de control, modificabilă mult mai ușor în comparație cu logica de tip cablat.

10.1.2.2.3. *Logica de interfață cu UM*

Blocul de interfață cu UM, dat în figura 10.7, cuprinde emițătorii și receptorii de interfață cu UM, multiplexorul pentru selecția informației ZRSM00-15* care se scrie în UM, multiplexorul pentru selecția adresei-memorie ZADR00-16, logica pentru poziționarea și ștergerea cererii de acces la memorie, logica pentru memorarea erorilor, registrele pentru memorarea diverselor informații necesare în timpul operațiilor de transfer. Blocul este controlat de SCM prin intermediul *bus*-urilor de control, date și condiționări. Se observă că microprogramul controlează selecția ZRSM00-15, ZADR00-16, poziționarea și ștergerea cererii de acces ZDAM, precum și încărcarea registrelor.

10.1.2.3. **Proiectarea structurii de control microprogramate**

10.1.2.3.1. *Structura hardware*

Ținînd cont că mașina microprogramată discutată aici implementează un algoritm de adaptare de interfețe, fiind deci un dispozitiv tipic de control (v. § 7.4), funcțiunile primitive ale ei vor fi mai mult de tip testare și pozi-

* Semnalele care încep cu Z sînt semnale TTL din CS înainte/după emițătorii/receptorii de interfață cu UM.

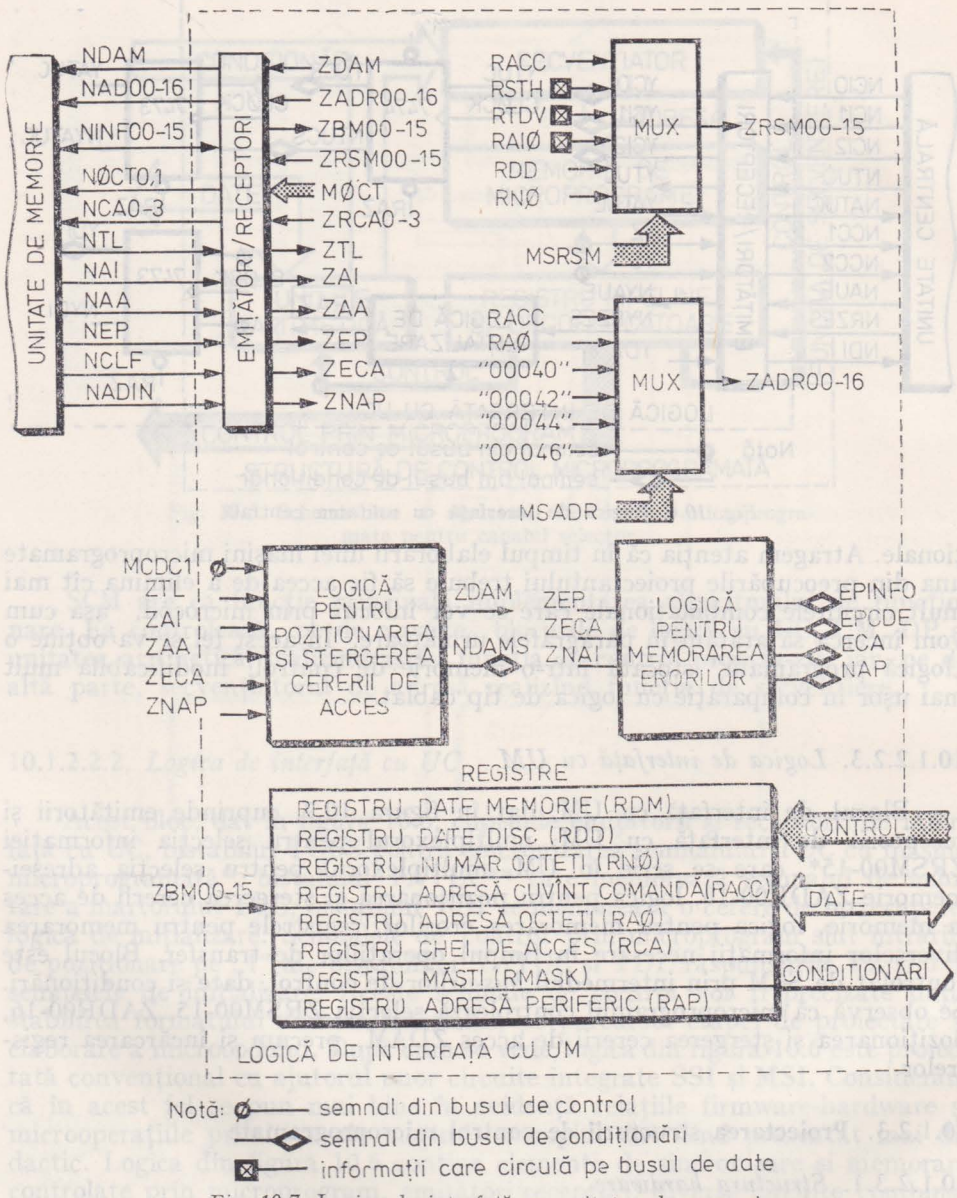


Fig. 10.7. Logica de interfață cu unitatea de memorie

ționare și mai puțin de tip aritmetic. Aceasta conduce la concluzia că SCM trebuie să implementeze cît mai bine funcția de secvențiere, accentuînd modul de testare a condițiilor și că unitatea aritmetică-logică asociată SCM poate fi foarte simplă. Plecînd de la aceste premize, secvențiatorul de microprogram se poate realiza cu circuite 2911 și 2909 și multiplexoare de selecție a indica-

torilor de condiție testați. Ieșirile acestor multiplexoare se suprapun prin SAU LOGIC peste cei mai puțin semnificativi doi biți ai adresei de microinstrucțiune generate de secvențiatoarele *bit-slice* (fig. 10.8). După cum se vede, pentru CS descris aici, adresa de microinstrucțiune are 12 biți, ceea ce permite adresarea a maximum 4K microinstrucțiuni. Multiplexoarele pot selecta simultan doi indicatori de condiție asigurînd în acest fel o testare multidirecțională rapidă într-un singur ciclu de microinstrucțiune. Cîmpul constantă, MK, din microinstrucțiune este utilizat sub controlul cîmpului de adresă următoare, MCAU, fie la intrările directe D ale secvențiatorului, fie pentru încărcarea registrului AR din interiorul circuitelor 2911 și 2909. Condițiile testate se selectează cu ajutorul cîmpurilor MSCA și MSCB. Funcția de secvențiere a SCM este deci controlată prin microprogram cu ajutorul cîmpurilor MCAU, MK, MSCA, MSCB.

Unitatea aritmetică, specifică majorității mașinilor microprogramate, este realizată aici cu circuite integrate MSI, conform schemei-bloc din figura 10.9. După cum se vede, este o unitate specializată fiind compusă dintr-o memorie de lucru, un multiplexor cu ajutorul căruia se selectează informațiile care se scriu în această memorie, un sumator și un comparator. Memoria de lucru cu două *port*-uri de ieșire conține 8 registre necesare în diverse operații executate cu unitatea aritmetică. Multiplexorul servește atît pentru selecția informației care se scrie în registrele de lucru, cît și pentru realizarea operațiilor de deplasare între două registre, prin selectarea unei reacții corespunzătoare care concatenează biții celor două ieșiri ale memoriei. Ieșirile memoriei reprezintă intrările stînga și dreapta ale sumatorului/comparatorului. Ieșirea sumatorului se poate rescrie într-un registru de lucru. Ieșirile comparatorului, EGAL și MM, semnalizează IS = ID, respectiv, IS > ID. Unitatea aritmetică este controlată prin microprogram, conform structurii sale, de cîmpurile MSIRL, MIS, MID, MWE și MCOMP. Cîmpul MSIRL selectează informația care se scrie în registrele de lucru. MIS și MID selectează registrele de lucru la cele două *port*-uri de ieșire ale memoriei, iar MWE validează operația de scriere în registrul de lucru selectat. Operațiile de comparare sînt controlate de cîmpul MCOMP.

10.1.2.3.2. *Formatul microinstrucțiunii*

În această etapă, avînd la dispoziție schema-bloc, proiectantul trebuie să definească într-un mod cît mai complet cîmpurile microinstrucțiunii cu scopul de a asigura controlul prin microprogram al întregii mașini.

Cîmpurile microinstrucțiunii trebuie să asigure controlul flexibil al tuturor resurselor hardware ale mașinii grupate în blocuri funcționale. În cazul nostru putem împărți cîmpurile microinstrucțiunii în cîmpuri pentru controlul secvențiatorului, al unității aritmetice, al celor trei blocuri de logică cu UM, UC și DP. De asemenea, se poate prevedea un cîmp pentru controlul ceasului mașinii.

Pentru mărirea vitezei de lucru se va utiliza o tehnică simplă de codificare a microinstrucțiunii, și anume cea pe un singur nivel (v. §7.2.2.2.). Microinstrucțiunea va fi de tip orizontal asigurînd controlul simultan al tuturor

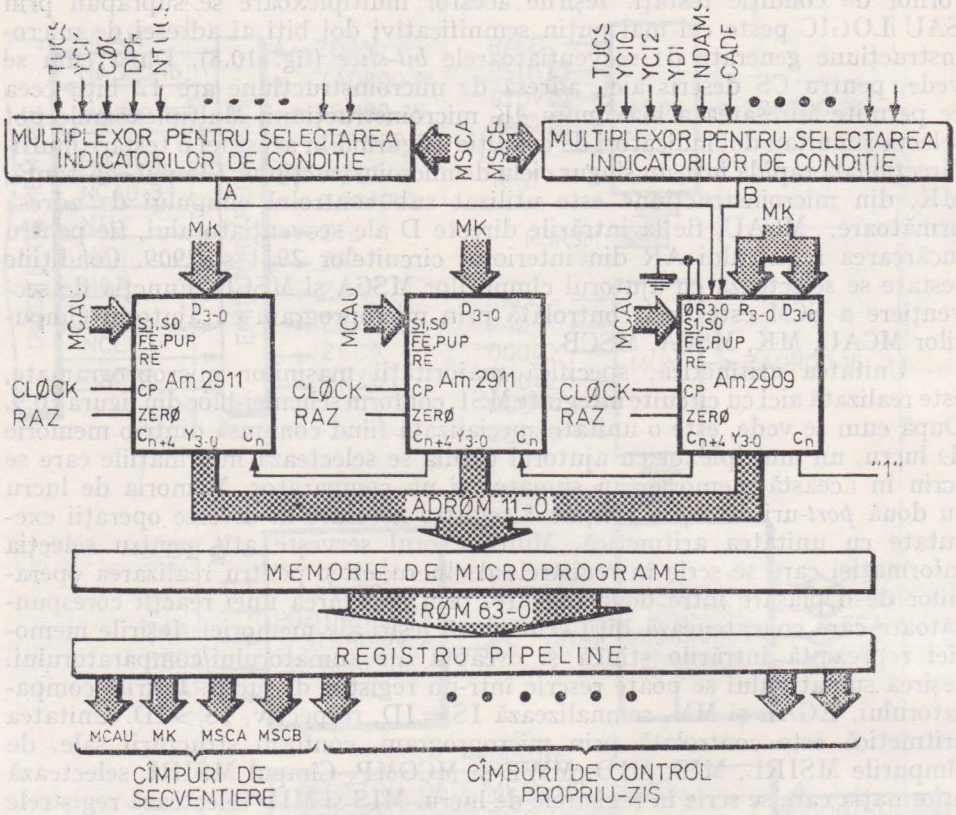


Fig. 10.8. Schema detaliată a structurii de control microprogramate pentru canalul selector

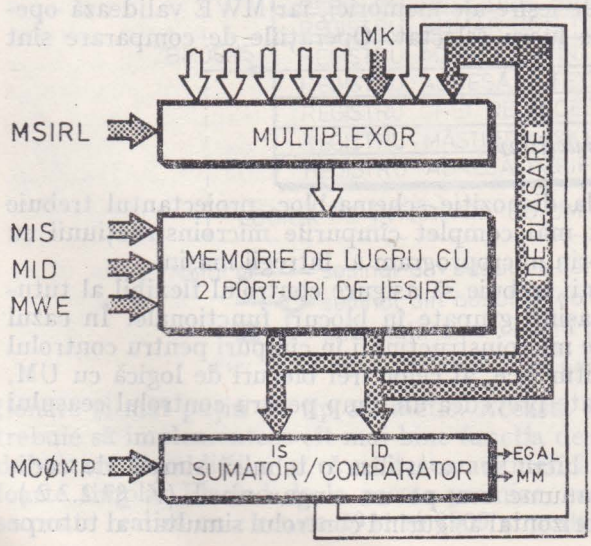


Fig. 10.9. Schema-bloc a unității aritmetice

resurselor hardware din mașină. Singura suprapunere este utilizarea cîmpului de constantă MK în determinarea adresei următoare și în operațiile unității aritmetice.

Așa cum s-a spus în §7.2.1, controlul prin microprogram se referă la controlul adresei următoare, funcția de secvențiere, și la controlul microoperațiilor realizate în mașină.

Conform schemei din figura 10.8 controlul adresei următoare de microinstrucțiune se face cu ajutorul cîmpurilor:

MCAU — controlul secvențiatorelor Am 2911 și Am 2909;
MK — cîmpul de constantă din microinstrucțiune;
MSCA — selecția condiției testate A;
MSCB — selecția condiției testate B.

Aceste cîmpuri determină funcționarea secvențiatorului de adresă de microprogram din cadrul SCM.

Unitatea aritmetică din SCM este controlată de cîmpurile:

MSIRL — selecția intrării în registrele de lucru;
MIS — intrarea stînga sumator;
MID — intrarea dreapta sumator;
MWE — validare-scriere în registrele de lucru;
MCOMP — controlul comparatorului.

Cîmpul de constantă MK este utilizat și pentru încărcarea registrelor de lucru.

Microoperațiile executate de blocurile de logică cu UM, UC, DP sînt controlate prin intermediul cîmpurilor:

MOCT — selecția octetului transferat cu UM;
MSADR — selecția adresei UM;
MSRSM — selecția registrului care se scrie în UM;
MZS — controlul indicatorilor de condiție Z, S;
MDP — controlul blocului de logică cu DP;
MCDA, MCDB, MCDC — control direct, poziționări, ștergeri, încărcări, în cele trei blocuri de logică.

Formatul microinstrucțiunii este dat în figura 10.10.

Menționăm că stabilirea formatului microinstrucțiunii urmează, ca orice proiect ingineresc, un proces iterativ. Proiectantul încearcă în prima etapă să stabilească principalele cîmpuri de control. În continuare, pe parcursul ela-

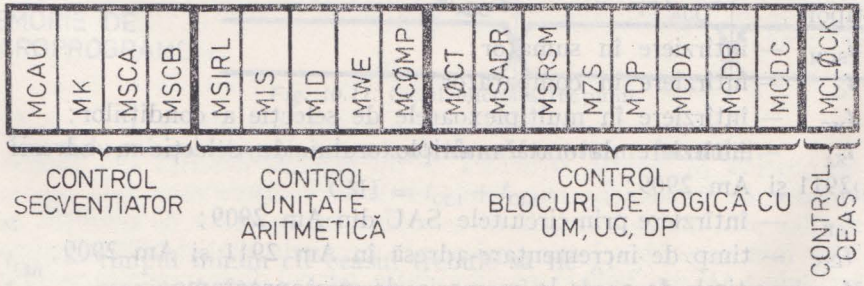


Fig. 10.10. Formatul microinstrucțiunii

borării, definitivării proiectelor firmware și hardware se definesc codurile mnemonice stabilindu-se mărimea cîmpurilor și se corectează formatul microinstrucțiunii adăugînd sau eliminînd unele cîmpuri. Scopurile activității de proiectare a formatului microinstrucțiunii sînt minimizarea memoriei de microprograme și a circuitelor aferente, obținerea unui control microprogramat cît mai flexibil, mărirea vitezei de lucru a mașinii.

10.1.2.3.3. *Ciclul microinstrucțiunii*

O altă etapă ce trebuie parcursă în elaborarea unei mașini microprogramate este determinarea ciclului de microinstrucțiune, CMI, care are ca scop găsirea unei soluții optime pentru asigurarea unei viteze de lucru maxime.

Într-o mașină microprogramată ciclurile de microinstrucțiune sînt determinate de întîrzierile componentelor de-a lungul diferitelor căi de control. Într-o SCM de tip *pipeline* la începutul fiecărui CMI ieșirile memoriei de microprograme sînt încărcate în registrul de microinstrucțiune. Acum încep toate calculele de timp pentru diverse căi de control. Inspecția vizuală a schemei-bloc nu va conduce întotdeauna la găsirea celor mai importante căi de control din punctul de vedere al calculului CMI. De aceea pentru o proiectare foarte îngrijită trebuie calculate toate căile de control ținînd cont de schema-bloc și formatul microinstrucțiunii. După stabilirea acestor calcule proiectantul hardware va preciza, în funcție de restricțiile de viteză și complexitate, ciclul microinstrucțiunii. Așa cum s-a spus în §7.2.2.3, CMI poate fi de tip monofază, polifază, variabil monofază sau variabil polifază.

În figura 10.11 se dau principalele întîrzieri corespunzătoare resurselor hardware din mașina microprogramată discutată aici. Acestea sînt:

t_{SU} — întîrziere datorată încărcării microinstrucțiunii în registrul *pipeline* sau poziționării și încărcării diverșilor *bistabili*, respectiv registre din mașină, așa-numitul timp de *set-up*;

t_D — întîrziere în decodificatoarele de microinstrucțiune;

t_{MUX1} — întîrziere în multiplexoarele pentru selecția ZSRSM, ZADR;

$t_{M1,2}$ — întîrzieri în multiplexoarele de selecție a informației care se va scrie în memoria de lucru;

t_{RL} — întîrziere datorată selectării registrelor de lucru din memoria multiport;

t_{SUM} — întîrziere în sumator;

t_C — întîrziere în comparator;

t_{MC} — întîrziere în multiplexoarele de selecție a condițiilor;

t_{SA} — întîrziere datorată multiplexorului de selecție a adresei din Am 2911 și Am 2909;

t_{OR} — întîrziere prin circuitele SAU din Am 2909;

t_{INC} — timp de incrementare-adresă în Am 2911 și Am 2909;

t_{ACC} — timp de acces la memoria de microprograme;

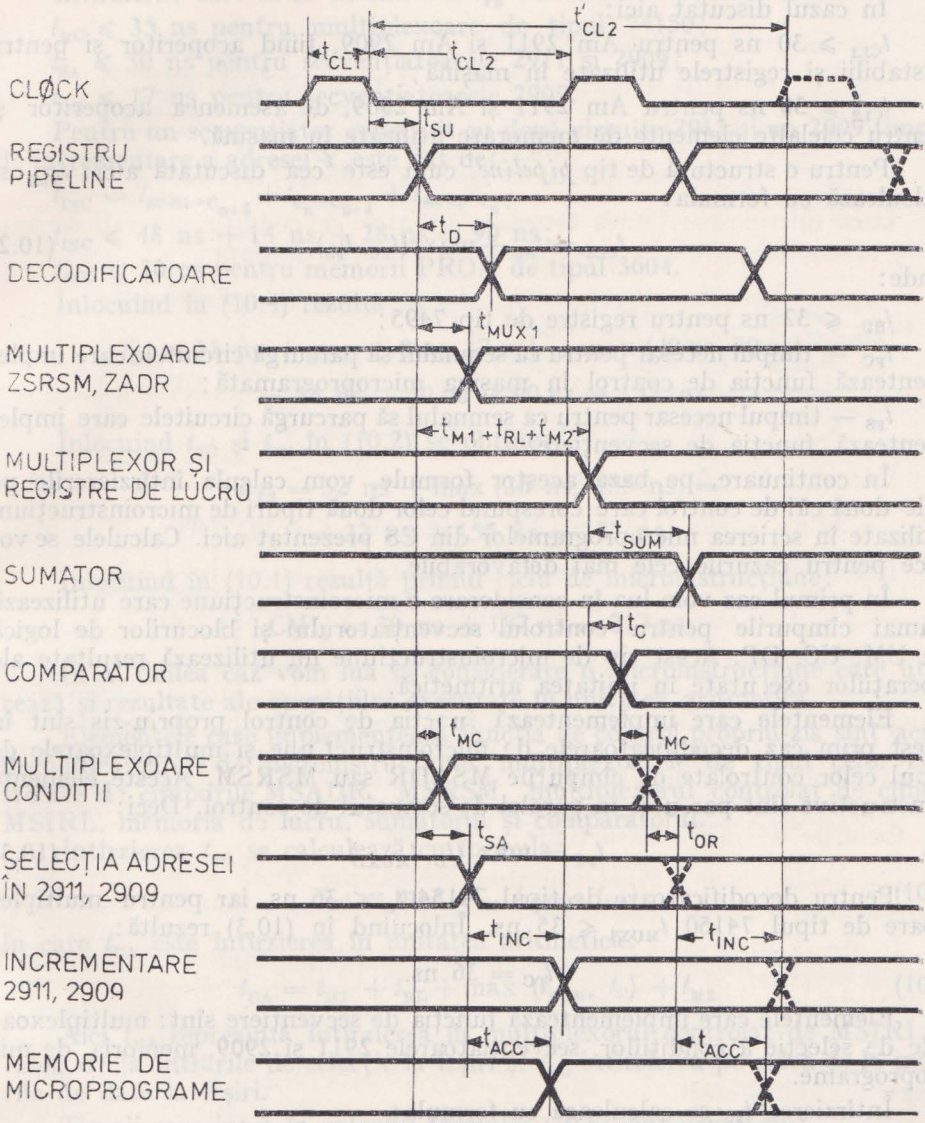


Fig. 10.11. Ciclul microinstrucțiunii

Un ciclu-microinstrucțiune se calculează conform formulei:

$$\text{CMI} = t_{CL1} + t_{CL2}, \tag{10.1}$$

unde:

t_{CL1} — timpul minim cît ceasul trebuie să fie „1";

t_{CL2} — maximul între întîrzierea pe calea de control considerată și timpul minim cît ceasul trebuie să fie „0".

În cazul discutat aici:

$t_{CL1} \geqslant 30$ ns pentru Am 2911 și Am 2909, fiind acoperitor și pentru bistabilii și registrele utilizate în mașină;

$t_{CL2} \geqslant 30$ ns pentru Am 2911 și Am 2909, de asemenea acoperitor și pentru celelalte elemente de memorare utilizate în mașină.

Pentru o structură de tip *pipeline*, cum este cea discutată aici; t_{CL2} se calculează cu formula:

$$t_{CL2} = t_{SU} + \max\ (t_{FC}, t_{FS}), \tag{10.2}$$

unde:

$t_{SU} \leqslant 32$ ns pentru registre de tip 7495;

t_{FC} — timpul necesar pentru ca semnalul să parcurgă circuitele care implementează funcția de control în mașina microprogramată;

t_{FS} — timpul necesar pentru ca semnalul să parcurgă circuitele care implementează funcția de secvențiere.

În continuare, pe baza acestor formule, vom calcula întîrzierile pe cele două căi de control care corespund celor două tipuri de microinstrucțiuni utilizate în scrierea microprogramelor din CS prezentat aici. Calculele se vor face pentru cazurile cele mai defavorabile.

În primul caz vom lua în considerare o microinstrucțiune care utilizează numai cîmpurile pentru controlul secvențiatorului și blocurilor de logică cu UM, UC, DP. Acest tip de microinstrucțiune nu utilizează rezultate ale operațiilor executate în unitatea aritmetică.

Elementele care implementează funcția de control propriu-zis sînt în acest prim caz decodificatoarele de microinstrucțiune și multiplexoarele de tipul celor controlate de cîmpurile MSADR sau MSRSM. Aceste elemente constructive sînt parcurse în paralel de semnalul de control. Deci:

$$t_{FC} = \max\ (t_D, t_{MUX1}). \tag{10.3}$$

Pentru decodificatoare de tipul 74154 $t_D \leqslant 36$ ns, iar pentru multiplexoare de tipul 74150 $t_{MUX1} \leqslant 35$ ns. Înlocuind în (10.3) rezultă:

$$t_{FC} = 36 \text{ ns}.$$

Elementele care implementează funcția de secvențiere sînt: multiplexoarele de selecție a condițiilor, secvențiatoarele 2911 și 2909, memoria de microprograme.

Întîrzierea t_{FS} se calculează cu formula:

$$t_{FS} = t_{MC} + \max\ (t_{SA}, t_{OR}) + \max\ (t_{INC}, t_{ACC}). \tag{10.4}$$

Se observă că, pentru structura de tip *pipeline* descrisă aici, întîrzierea t_{FS} pe calea de implementare a funcției de secvențiere este dată de suma întîrzierilor prin multiplexoarele de selecție a indicatorilor de condiție, maximul între întîrzierea de selecție a adresei în circuitele 2911 și 2909 și întîrzierea în circuitele SAU din 2909 și maximul între timpul de incrementare-adresă și timpul de acces la memoria de microprograme.

Întîrzierile care intră în calculul t_{FS} sînt date în continuare:

$t_{MC} \leqslant 35$ ns pentru multiplexoare de tipul 74150;

$t_{SA} \leqslant 30$ ns pentru secvențiatoarele 2911 și 2909;

$t_{OR} \leqslant 17$ ns pentru secvențiatoarele 2909.

Pentru un secvențiator construit cu două circuite 2911 și un 2909 timpul de incrementare a adresei Y este dat de:

$t_{INC} = t_{S0,S1 \to C_{n+4}} + t_{C_n \to C_{n+4}} + t_{\text{set-up } C_n}$;

$t_{INC} \leqslant 48$ ns $+ 14$ ns $+ 28$ ns $= 90$ ns;

$t_{ACC} \leqslant 70$ ns pentru memorii PROM de tipul 3604.

Înlocuind în (10.4) rezultă:

$$t_{FS} = 35 \text{ ns} + \max (30 \text{ ns}, 17 \text{ ns}) + \max (90 \text{ ns}, 70 \text{ ns}) =$$

$$= 35 \text{ ns} + 30 \text{ ns} + 90 \text{ ns} = 155 \text{ ns}.$$

Înlocuind t_{FC} și t_{FS} în (10.2) se obține:

$$t_{CI2} = 32 \text{ ns} + \max (36 \text{ ns}, 155 \text{ ns}) =$$

$$= 32 \text{ ns} + 155 \text{ ns} = 187 \text{ ns}.$$

Înlocuind în (10.1) rezultă primul ciclu de microinstrucțiune:

$$CMI_1 = 30 \text{ ns} + 187 \text{ ns} = 217 \text{ ns}.$$

În al doilea caz vom lua în considerare o microinstrucțiune care utilizează și rezultate ale operațiilor executate în unitatea aritmetică.

Elementele care implementează funcția de control propriu-zis sînt acum decodificatoarele de microinstrucțiune, multiplexoarele de tipul celor controlate de cîmpurile MSADR, MSRSM, multiplexorul controlat de cîmpul MSIRL, memoria de lucru, sumatorul și comparatorul.

Întîrzierea t_{FC} se calculează cu formula:

$$t_{FC} = \max (t_D, t_{MUX1}, t_{UA}), \tag{10.5}$$

în care t_{UA} este întîrzierea în unitatea aritmetică:

$$t_{UA} = t_{M1} + t_{RL} + \max (t_{SUM}, t_C) + t_{M2} \tag{10.6}$$

Aici t_{M1} reprezintă întîrzierea în multiplexorul controlat de MSIRL pe calea de la intrările de selecție la ieșiri și t_{M2} întîrzierea pe calea de la intrările de date la ieșiri.

Timpii care intră în calculul formulei (10.6) sînt următorii:

$t_{M1} \leqslant 18$ ns pentru multiplexoare 74S151;

$t_{M2} \leqslant 12$ ns pentru multiplexoare 74S151;

$t_{RL} \leqslant 55$ ns pentru operații de citire/scriere în memoria multiport de tipul 82S112;

$t_{SUM} \leqslant 22{,}5$ ns pentru un sumator realizat cu două circuite 74S181;

$t_C \leqslant 28{,}5$ ns pentru un comparator construit cu două circuite 74S85.

Înlocuind în (10.6) rezultă:

$$t_{UA} = 18 \text{ ns} + 55 \text{ ns} + \max (22{,}5 \text{ ns}, 28{,}5 \text{ ns}) + 12 \text{ ns} =$$

$$= 18 \text{ ns} + 55 \text{ ns} + 28{,}5 \text{ ns} + 12 \text{ ns} = 113{,}5 \text{ ns}.$$

Substituind în (10.5) se găsește:

$$t_{FC} = \max (36 \text{ ns}, 35 \text{ ns}, 113{,}5 \text{ ns}) = 113{,}5 \text{ ns}.$$

Dacă microinstrucțiunea testează indicatori de condiție generați de unitatea aritmetică, atunci t_{FS} se calculează în felul următor:

$$t_{FS} = \max (t_{SA}, t_{FC} + t_{MC} + t_{OR}) + \max (t_{INC}, t_{ACC}).$$

Înlocuind se obține:

$$t_{FS} = \max (30 \text{ ns}, 113{,}5 \text{ ns} + 35 \text{ ns} + 17 \text{ ns}) + \max (90 \text{ ns},$$

$$70 \text{ ns}) = 165{,}5 \text{ ns} + 90 \text{ ns} = 255{,}5 \text{ ns}.$$

Înlocuind în (10.2) rezultă:

$$t'_{CL2} = 32 \text{ ns} + \max (113{,}5 \text{ ns}, 255{,}5 \text{ ns}) = 32 \text{ ns} + 255{,}5 \text{ ns} = 287{,}5 \text{ ns}.$$

Înlocuind în (10.1) se obține al doilea ciclu de microinstrucțiune:

$$CMI_2 = t_{CL1} + t'_{CL2} = 30 \text{ ns} + 287{,}5 \text{ ns} = 317{,}5 \text{ ns}.$$

Corespunzător celor două tipuri de microinstrucțiune utilizate în mașina microprogramată descrisă aici am obținut două cicluri diferite de microinstrucțiune, CMI_1 și CMI_2. Pentru mărirea vitezei de lucru, pe baza acestor cicluri, se poate utiliza un ceas de microinstrucțiune monofază variabil cu două perioade. Cele două perioade vor trebui să aibă ca multiplu comun o cuantă, perioada unei frecvențe, ce se poate obține cu ajutorul unui oscilator.

Ciclurile de microinstrucțiune alese sînt:

$$CMI_1 \text{ (ales)} = 240 \text{ ns};$$
$$CMI_2 \text{ (ales)} = 320 \text{ ns}.$$

În figura 10.12 se dă o schemă de realizare a unui ceas-monofază variabil cu două perioade. Se observă că ceasul mașinii, CLOCK, se obține prin intermediul a două numărătoare încărcate la începutul fiecărui ciclu de microinstrucțiune cu numere diferite. Impulsul următor de ceas rezultă prin selectarea cu ajutorul cîmpului de microinstrucțiune MCLOCK a transportului corespunzător ciclului dorit.

10.1.3. PROIECT FIRMWARE

10.1.3.1. Organigrama generală a microprogramului

În această etapă trebuie elaborată o organigramă globală a microprogramului de control corespunzătoare algoritmului pe care este necesar să îl realizeze mașina. Pentru CS organigrama generală a microprogramului este

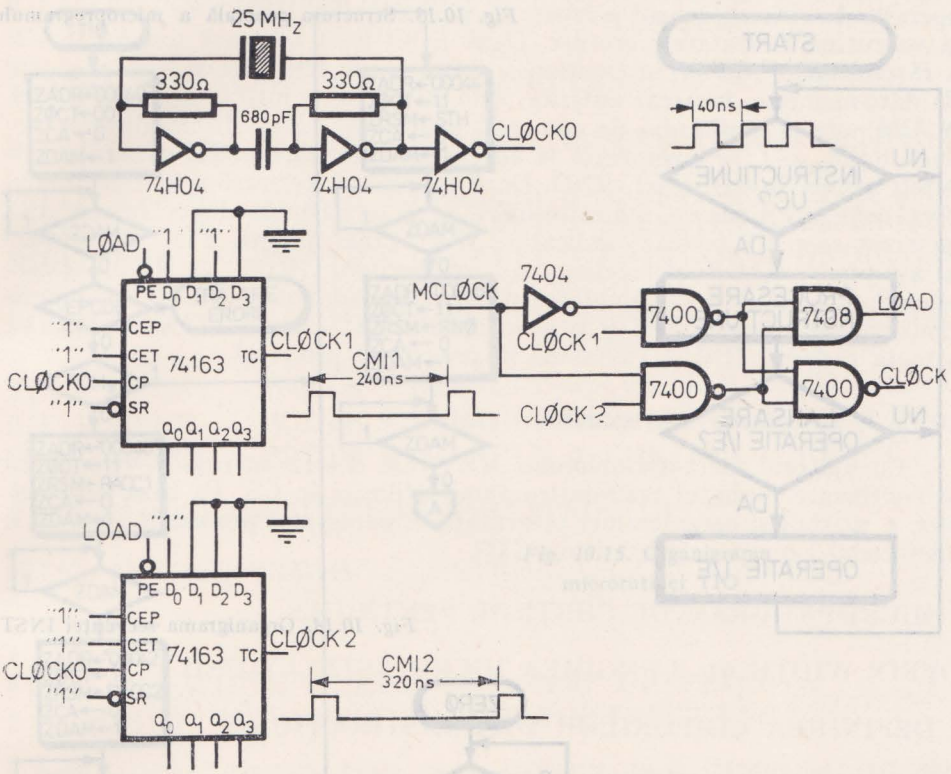


Fig. 10.12. Schema de realizare a unui ceas monofază cu două perioade

dată în figura 10.13. După inițializare mașina așteaptă o instrucțiune de la UC. În cazul unei instrucțiuni SIO reușite se va lansa execuția unei operații de intrare/ieșire.

În continuare, după elaborarea acestei organigrame globale, proiectantul general al microprogramului poate detalia printr-o abordare de tipul *top-down* fiecare element al organigramei stabilind modulele microcodului și interfețele între ele.

Urmează apoi elaborarea organigramelor detaliate. În această etapă microprogramatorul trebuie să utilizeze cît mai bine formatul microinstrucțiunii și, implicit, resursele hardware ale mașinii. Stabilirea organigramelor va conduce la detalierea cîmpurilor microinstrucțiunii, uneori la modificări de format și/sau modificări hardware.

10.1.3.2. Organigrama unei microrutine

Vom prezenta în continuare o rutină care implementează prin microcod răspunsul mașinii microprogramate prezentate aici, canalul selector, la instrucțiunea TIO lansată de unitatea centrală a calculatorului.

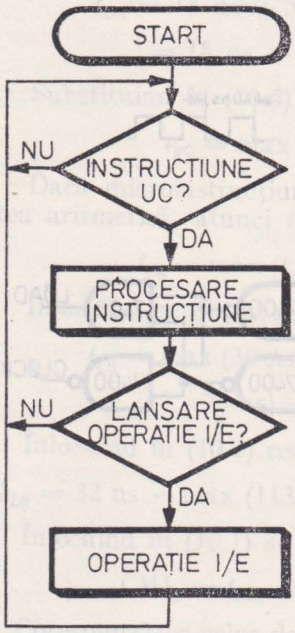
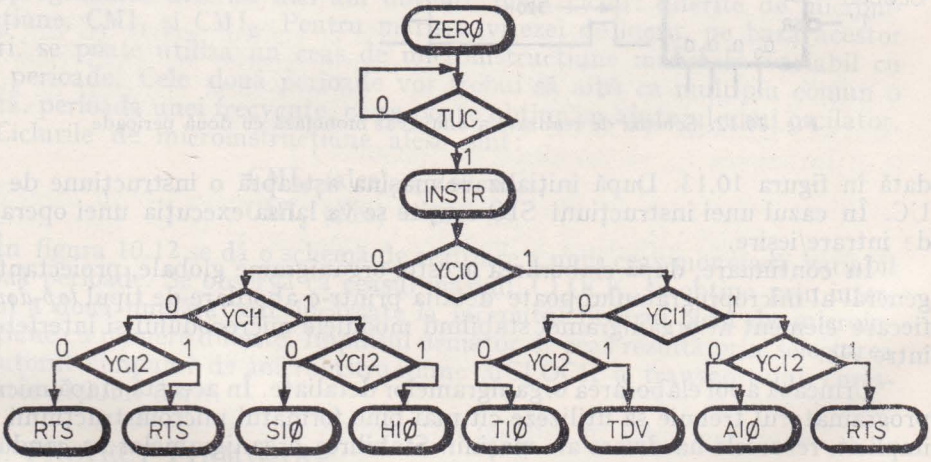


Fig. 10.13. Structura generală a microprogramului

Fig. 10.14. Organigrama secvenței INSTR



După inițializare CS așteaptă o instrucțiune de la UC testînd prezența semnalului de eșantionare TUC. Urmează secvența INSTR de testare a codurilor-instrucțiune care selectează rutina de tratare corespunzătoare. În figura 10.14 se dă organigrama secvenței INSTR.

Rutina TIO, răspunsul la instrucțiunea de test intrare/ieșire TIO, are organigrama din figura 10.15. După cum se vede, rutina începe printr-o citire din memoria calculatorului de la adresa 00040H. Dacă în timpul acestei

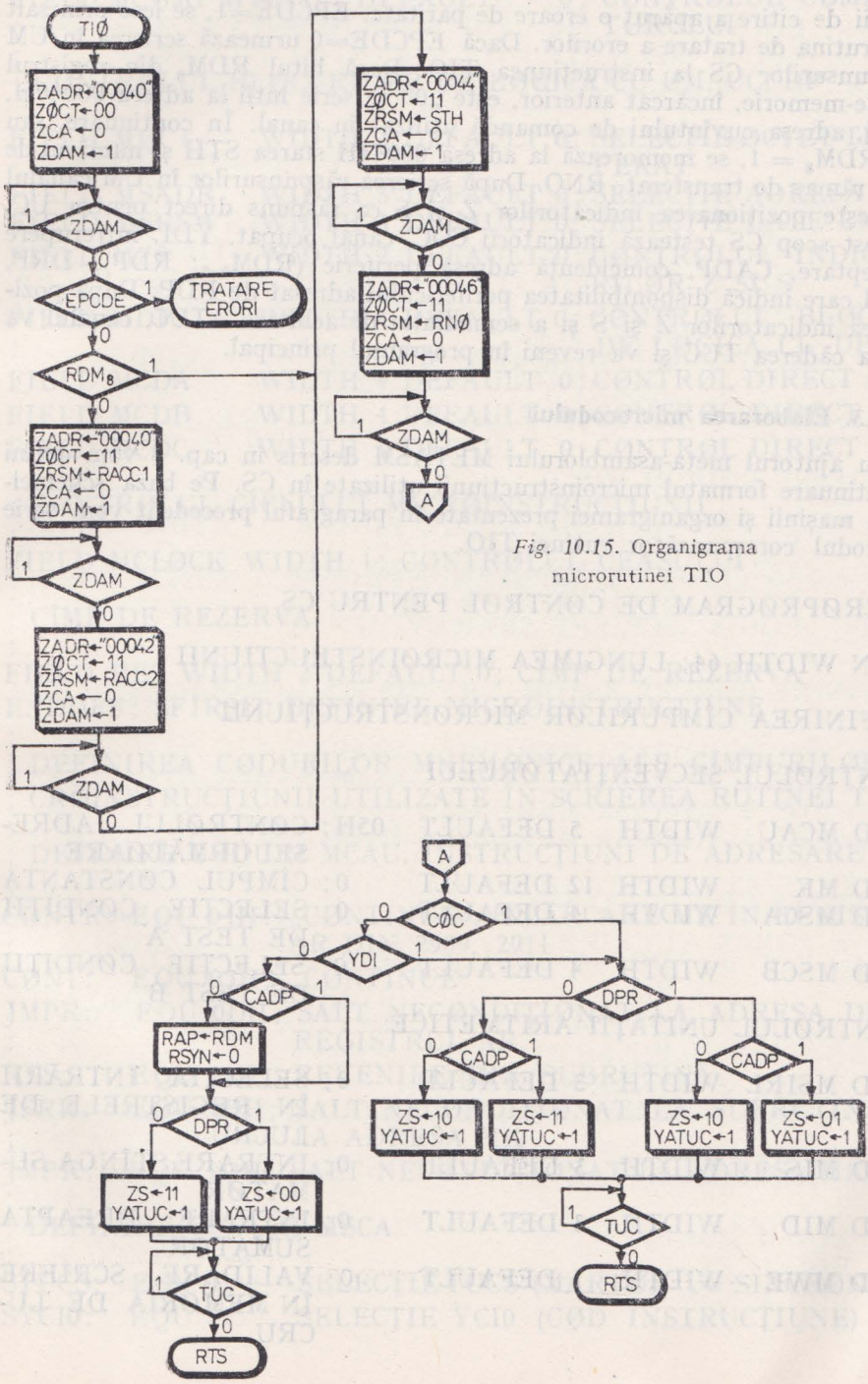


Fig. 10.15. Organigrama microrutinei TIO

operații de citire a apărut o eroare de paritate, EPCDE=1, se iese prin salt la subrutina de tratare a erorilor. Dacă EPCDE=0 urmează scrierea în UM a răspunsurilor CS la instrucțiunea TIO. Dacă bitul RDM_8 din registrul de date-memorie, încărcat anterior, este „0" se scrie întîi la adresa 00040H, în UM, adresa cuvîntului de comandă utilizat în canal. În continuare, sau dacă $RDM_8 = 1$, se memorează la adresa 00044H starea STH și numărul de octeți rămas de transferat, RNO. După scrierea răspunsurilor în UM canalul pregătește poziționarea indicatorilor Z și S ca răspuns direct pentru UC. În acest scop CS testează indicatorii COC, canal ocupat, YDI, întrerupere în așteptare, CADP, coincidență adresă periferic (RDM_{0-7}: RDP), DRP, semnal care indică disponibilitatea perifericului adresat de RDP. După poziționarea indicatorilor Z și S și a semnalului de achitare ATUC, canalul va aștepta căderea TUC și va reveni în programul principal.

10.1.3.3. Elaborarea microcodului

Cu ajutorul meta-asamblorului METASM descris în cap. 8 vom defini în continuare formatul microinstrucțiunii utilizate în CS. Pe baza schemei-bloc a mașinii și organigramei prezentate în paragraful precedent vom scrie microcodul corespunzător rutinei TIO.

```
;
; MICRØPRØGRAM DE CØNTRØL PENTRU CS
;
DEFIN WIDTH 64; LUNGIMEA MICRØINSTRUCȚIUNII
;
; DEFINIREA CÎMPURILØR MICRØNSTRUCȚIUNII
;
; CØNTRØLUL SECVENȚIATØRULUI
;
FIELD MCAU   WIDTH  5 DEFAULT  05H; CØNTRØLUL ADRE-
;                                   SEI URMĂTØARE
FIELD MK     WIDTH 12 DEFAULT    0; CÎMPUL CØNSTANTA
FIELD MSCA   WIDTH  4 DEFAULT    0; SELECȚIE CØNDIȚII
;                                   DE TEST A
FIELD MSCB   WIDTH  4 DEFAULT    0; SELECȚIE CØNDIȚII
;                                   DE TEST B
; CØNTRØLUL UNITĂȚII ARITMETICE
;
FIELD MSIRL  WIDTH  3 DEFAULT    0; SELECȚIA INTRĂRII
;                                   ÎN REGISTRELE DE
;                                   LUCRU
FIELD MIS    WIDTH  3 DEFAULT    0; INTRARE STÎNGA SU-
;                                   MATØR
FIELD MID    WIDTH  3 DEFAULT    0; INTRARE DREAPTA
;                                   SUMATØR
FIELD MWE    WIDTH  1 DEFAULT    0; VALIDARE SCRIERE
;                                   ÎN MEMØRIA DE LU-
;                                   CRU
```

```
FIELD MCØMP WIDTH 2 DEFAULT    0; CØNTRØLUL CØMPARA-
;                                 TØRULUI
;
; CØNTRØLUL BLØCURILØR DE LØGICA CU UM, UC, DP
;
FIELD MØCT      WIDTH 2 DEFAULT 0; SELECȚIE ØCTET TRANS-
;                                  FERAT
FIELD MSADR     WIDTH 3 DEFAULT 0; SELECȚIE ADRESA UM
FIELD MSRSM     WIDTH 3 DEFAULT 0; SELECȚIE DATE UM
FIELD MZS       WIDTH 2 DEFAULT 0; CØNTRØLUL INDICATØ-
;                                  RILØR Z ȘI S
FIELD MDP       WIDTH 4 DEFAULT 0; CØNTRØLUL BLØCULUI
;                                  DE LØGICA CU DP
FIELD MCDA      WIDTH 4 DEFAULT 0; CØNTRØL DIRECT A
FIELD MCDB      WIDTH 4 DEFAULT 0; CØNTRØL DIRECT B
FIELD MCDC      WIDTH 3 DEFAULT 0; CØNTRØL DIRECT C
;
; CØNTRØLUL CICLULUI MICRØINSTRUCȚIUNII
;
FIELD MCLØCK WIDTH 1; CØNTRØLUL CEASULUI
;
; CÎMP DE REZERVA
;
FIELD REZ WIDTH 2 DEFAULT 0; CÎMP DE REZERVA
ENDDEF; SFÎRȘIT DEFINIRE MICRØINSTRUCȚIUNE
;
; DEFINIREA CØDURILØR MNEMØNICE ALE CÎMPURILØR MI-
; CRØINSTRUCȚIUNII UTILIZATE ÎN SCRIEREA RUTINEI TIØ
;
; DEFINIRE CØDURI MCAU, INSTRUCȚIUNI DE ADRESARE
;
CØNTR: EQU 04H; „CØNTINUE", INCARCARE MK IN REGISTRUL
;               AR DIN 2909, 2911
CØNT:  EQU 05H; „CØNTINUE"
JMPR:  EQU 0DH; SALT NECØNDIȚIØNAT LA ADRESA DIN
;               REGISTRUL AR
RTS:   EQU 11H; REVENIRE DIN SUBRUTINĂ
JSRK:  EQU 1BH; SALT NECØNDIȚIØNAT LA SUBRUTINA DE
;               LA ADRESA MK
JMPK:  EQU 1DH; SALT NECØNDIȚIØNAT LA ADRESA MK
;
; DEFINIRE CODURI MSCA
;
STUCS: EQU 1;   SELECȚIE TUCS (MARTØR UC SINCRØNIZAT)
SYCI0: EQU 2;   SELECȚIE YCI0 (CØD INSTRUCȚIUNE)
```

```
SCØC:    EQU 3 ;SELECȚIE CØC (CANAL ØCUPAT)
SCADP:   EQU 4 ;SELECȚIE CADP (CØINCIDENTA ADRESA DP)
SNTUCS:  EQU 5 ;SELECȚIE NTUCS (MARTØR UC NEGAT)
SEPCDE:  EQU 6 ;SELECȚIE EPCDE (ERØARE PARITATE)
;
; DEFINIRE CØDURI MSCB
;
STUCSB:  EQU 1 ;SELECȚIE TUCS (MARTØR UC)
SYCI1:   EQU 2 ;SELECȚIE YCI1 (CØD INSTRUCȚIUNE)
SYCI2:   EQU 3 ;SELECȚIE YCI2 (CØD INSTRUCȚIUNE)
SYDI:    EQU 4 ;SELECȚIE YDI (INTRERUPERE ÎN AȘTEPTARE)
SNDAMS:  EQU 5 ;SELECȚIE NDAMS (CERERE ACCES MEMØRIE
               ;ACHITATĂ)
SDPR:    EQU 6 ;SELECȚIE DPR (DISPØZITIV PERIFERIC
               ;„READY")
SRDM8:   EQU 7 ;SELECȚIE RDM8 (BITUL 8 DIN REGISTRUL RDM)
;
; DEFINIRE CØDURI MØCT
;
ØCT00:   EQU 0 ;CITIRE SEMICUVÎNT
ØCT01:   EQU 1 ;SCRIERE ØCTET DREAPTA
ØCT10:   EQU 2 ;SCRIERE ØCTET STÎNGA
ØCT11:   EQU 3 ;SCRIERE SEMICUVÎNT
;
; DEFINIRE CØDURI MSADR
;
SAD40: EQU 0 ;ADRESA MEMØRIE=00040H
SAD42: EQU 1 ;ADRESA MEMØRIE=00042H
SAD44: EQU 2 ;ADRESA MEMØRIE=00044H
SAD46: EQU 3 ;ADRESA MEMØRIE=00046H
;
; DEFINIRE CØDURI MSRSM
;
RACC1:  EQU 0 ;ADRESA CUVÎNT CØMANDA, SEMICUVÎNTUL
;              C.M.S.
RACC2:  EQU 1 ;ADRESA CUVÎNT CØMANDA, SEMICUVÎNTUL
;              C.M.P.S.
STH:    EQU 2 ;STARE SIØ/TIØ/HIØ
RNØ:    EQU 3 ;NUMĂR ØCTEȚI RĂMAS DE TRANSFERAT
;
; DEFINIRE CØDURI MZS
;
ZS00: EQU 0 ;ZS=00
ZS01: EQU 1 ;ZS=01
ZS10: EQU 2 ;ZS=10
ZS11: EQU 3 ;ZS=11
```

```
;
; DEFINIRE CØDURI MCDA
;
LRAP: EQU 1; INCARCARE RAP CU RDM0-7
FCAZ: EQU 2; FØRȚARE CHEI ACCES PE ZERØ
;
; DEFINIRE CØDURI MCDB
;
NRSYN: EQU 1; BISTABILUL DE RESINCRØNIZARE PERIFERIC
;              RSYN=0
;
; DEFINIRE CØDURI MCDC
;
ZDAM: EQU 1; CERERE ACCES MEMORIE, ZDAM=1
YATUC: EQU 2; ACHITARE TUC, YATUC=1
;
; DEFINIRE MICRØP-URI
;
MICRØP CØNTINUE ASSIGN MCAU=CØNT
MICRØP WAITR    ASSIGN MSCB=SNDAMS MØCT=ØCT00
MCDA=FCAZ;
MICRØP WAITW    ASSIGN MSCB=SNDAMS MØCT=ØCT11
MCDA=FCAZ;
;
;
; MICRØPRØGRAMUL DE CØNTRØL
;
;
MPRØG MPCS WIDTH 64;
;
AȘTEPTARE INSTRUCTIUNE UC
;
ZERØ: MCAU=JMPK MK=ZERØ MSCB=STUCSB; TESTARE TUC
      MCAU=JSRK MK=INSTR; APEL SUBRUTINA INSTR
      MCAU=JMPK MK=ZERØ; REVENIRE LA ZERØ
;
; TESTARE CØDURI INSTRUCȚIUNE
;
INSTR: MCAU=CØNT MSCA=SYCI0 MSCB=SYCI1; TESTARE
;YCI0, YCI1
YCI00: MCAU=JMPK MK=YCI000 MSCB=SYCI2; TESTARE
;YCI2=YCI0, 1=00
YCI01: MCAU=JMPK MK=YCI010 MSCB=SYCI2; TESTARE
;YCI2, YCI0, 1=01
YCI10: MCAU=JMPK MK=YCI100 MSCB=SYCI2; TESTARE
;YCI2, YCI0, 1=10
YCI11: MCAU=JMPK MK=YCI110 MSCB=SYCI2; TESTARE
;YCI2, YCI0, 1=11
```

```
;
; ÎNCEPUT TRATARE INSTRUCȚIUNI
;
YCI000: MCAU=RTS; CØD INSTRUCȚIUNE INCØRECT, NU SE
;ACHITA TUC
YCI001: MCAU=RTS; CØD INSTRUCȚIUNE INCØRECT, NU SE
;ACHITA TUC
YCI010: MCAU=JMPK MK=SIØ; INSTRUCȚIUNEA SIØ
YCI011: MCAU=JMPK MK=HIØ; INSTRUCȚIUNEA HIØ
YCI100: MCAU=JMPK MK=TIØ MØCT=ØCT00 MSADR=SAD40
MCDA=FCAZ MCDC=ZDAM; INSTRUCTIUNEA TIØ
YCI101: MCAU=JMPK MK=TDV; INSTRUCȚIUNEA TDV
YCI110: MCAU=JMPK MK=AIØ; INSTRUCȚIUNEA AIØ
YCI111: MCAU=RTS; CØD INSTRUCȚIUNE INCØRECT, NU SE
;ACHITA TUC
;
; TRATARE TIØ
;
TIØ: MCAU=JMPK MK=TIØ WAITR MSADR=SAD40; AȘTEPTARE
;CITIRE
;MCAU=JMPK MK=EPRD00 MSCA=SEPCDE MSCB=SRDM8;
;TESTARE ERØARE PARITATE ȘI BITUL 8 DIN RDM
TIØ1: MCAU=JMPK MK=TIØ1 WAITW MSADR=SAD40
MSRSM=RACC1; AȘTEPTARE CITIRE
MCAU=JMPK MK=TIØ2 MØCT=ØCT11 MSADR=SAD42
MSRSM=RACC2 MCDA=FCAZ MCDC=ZDAM ;SCRIERE RACC2 LA
;ADRESA 00042H
EPRD00: MCAU=JMPK MK=TIØ1 MØCT=ØCT11 MSADR=SAD40
MSRSM=RACC1 MCDA=FCAZ MCDC=ZDAM; SCRIERE RACC1
;LA ADRESA 00040H
EPRD01: MCAU=JMPK MK=TIØ3 MØCT=ØCT11 MSADR=SAD44
MSRSM=STH MCDA=FCAZ MCDC=ZDAM; SCRIERE STH LA
;ADRESA 00044H
EPRD10: MCAU=JMPK MK=ERØRI; ERØARE PARITATE LA
;CITIRE DIN UM
EPRD11: MCAU=JMPK MK=ERØRI; ERØARE PARITATE LA
;CITIRE DIN UM
TIØ2: MCAU=JMPK MK=TIØ2 WAITW MSADR=SAD42
MSRSM=RACC2; AȘTEPTARE SCRIERE
MCAU=CØNT MØCT=ØCT11 MSADR=SAD44 MSRSM=STH
MCDA=FCAZ MCDC=ZDAM; SCRIERE STH LA ADRESA 00044H
TIØ3: MCAU=JMPK MK=TIØ3 WAITW MSADR=SAD44
MSRSM=STH; AȘTEPTARE SCRIERE
MCAU=CØNT MØCT=ØCT11 MSADR=SAD46 MSRSM=RNØ
MCDA=FCAZ MCDC=ZDAM; SCRIERE RNØ LA ADRESA 00046H
```

```
TIØ4: MCAU=JMPK MK=TIØ4 WAITW MSADR=SAD46
MSRSM=RNØ; AȘTEPTARE SCRIERE
MCAU=JMPK MK=CØCDI MSCA=SCØC MSCB=SYDI; TESTARE
; CØC ȘI YDI
TIØ5: MCAU=JMPK MK=TIØ5 MSCB=STUCSB MZS=ZS00
; AȘTEPTARE TUC=0
RTS; REVENIRE IN PRØGRAMUL PRINCIPAL
CØCDI: MCAU=JMPK MK=TIØ6 MSCA=SCADP; CØC, YDI=00,
; TESTARE CADP
MCAU=JMPK MK=TIØ9 MSCA=SCADP MSCB=DPR; CØC, YDI=01,
; TESTARE; CADP SI DPR
MCAU=JMPK MK=TIØ9 MSCA=SCADP MSCB=DPR; CØC, YDI=10,
; TESTARE CADP SI DPR
MCAU=JMPK MK=TIØ9 MSCA=SCADP MSCB=DPR; CØC,
; YDI=11, TESTARE CADP ȘI DPR
TIØ6: MCAU=CØNT MCDA=LRAP MCDB=NRSYN;
; SINCRØNIZAZARE CU UN NØU DP
MCAU=JMPK MK=TIØ7 MSCB=SDPR; TESTARE DPR
MCAU=JMPK MK=TIØ7 MSCB=SDPR;
CØNTINUE; CADRAJ LA ADRESA XX...X00B
TIØ7: MCAU=JMPK MK=TIØ8 MZS=ZS11 MCDC=YATUC;
; ACHITARE TUC
MCAU=JMPK MK=TIØ5 MZS=ZS00 MCDC=YATUC; ACHITARE
; TUC
TIØ8: MCAU=JMPK MK=TIØ8 MSCB=STUCSB MZS=ZS11;
; ASTEPTARE TUC=0
RTS; REVENIRE IN PRØGRAMUL PRINCIPAL
TIØ9: MCAU=JMPK MK=TIØ10 MZS=ZS10 MCDC=YATUC;
; ACHITARE TUC
MCAU=JMPK MK=TIØ8 MCDC=YATUC MZS=ZS11; ACHITARE
; TUC
MCAU=JMPK MK=TIØ10 MZS=ZS10 MCDC=YATUC; ACHITARE
; TUC
MCAU=JMPK MK=TIØ11 MZS=ZS01 MCDC=YATUC; ACHITARE
; TUC
TIØ10: MCAU=JMPK MK=TIØ10 MSCB=STUCSB MZS=ZS10;
; AȘTEPTARE TUC=0
RTS; REVENIRE IN PRØGRAMUL PRINCIPAL
TIØ11: MCAU=JMPK MK=TIØ11 MSCB=STUCSB MZS=ZS01;
; AȘTEPTARE TUC=0
RTS; REVENIRE IN PRØGRAMUL PRINCIPAL
. . . . . .
END;
```

10.2. O UNITATE CENTRALĂ MICROPROGRAMATĂ CONSTRUITĂ CU CIRCUITE DIN FAMILIA Am 2900

10.2.1. SCHEMA-BLOC

Se știe că majoritatea calculatoarelor moderne au la bază conceptul de „calculator cu program stocat" introdus de matematicianul *John von Neumann*.

Un calculator cu program stocat se poate defini ca o mașină capabilă să manipuleze informații conform unor reguli prestabilite, instrucțiuni, avînd programul, o colecție de instrucțiuni, și datele stocate într-o memorie. Structura generală a unui calculator cu program stocat este dată în figura 10.16.

Una din aplicațiile cele mai cunoscute ale microprocesoarelor *bit-slice* este implementarea funcției de control în unități centrale de calculatoare (v. § 7.4). În acest tip de aplicație mașina microprogramată trebuie în principal să extragă din memorie și să interpreteze instrucțiunile-mașină. În figura 10.17 este dată organigrama generală de control pentru o unitate centrală de calculator indicîndu-se operațiile mari pe care trebuie să le execute mașina.

În continuare vom descrie o unitate centrală microprogramată realizată cu circuite LSI din familia Am 2900 [2].

Schema-bloc, principalele căi de date și control ale acestei unități centrale pe 16 biți sînt date în figura 10.18. Unitatea centrală se compune dintr-o SCM care adresează o memorie de microprograme, o unitate aritmetică-logică realizată cu patru microprocesoare *bit-slice* de tipul Am 2903, un circuit Am 2904 pentru controlul operațiilor de deplasare și al indicatorilor de condiție, un registru de intrare-date, un registru de ieșire-date, un registru de adresă și un registru pentru controlul operațiilor de I/E. Mașina comunică cu exteriorul prin intermediul a trei *bus*-uri: *bus*-ul de date, *bus*-ul de adresă și *bus*-ul de I/E.

Funcționarea circuitelor LSI utilizate pentru realizarea acestei unități centrale a fost descrisă în cap. 9. În general vom insista, atunci cînd este cazul, numai asupra unor particularități legate de interconectarea acestor circuite.

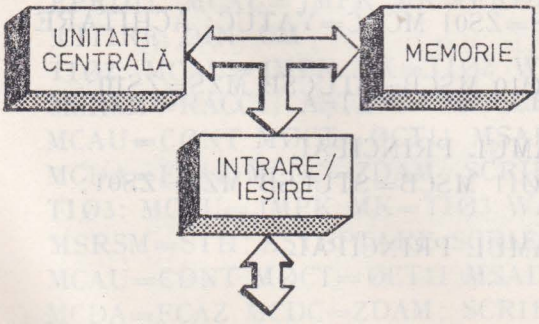


Fig. 10.16. Structura generală a unui calculator cu program stocat

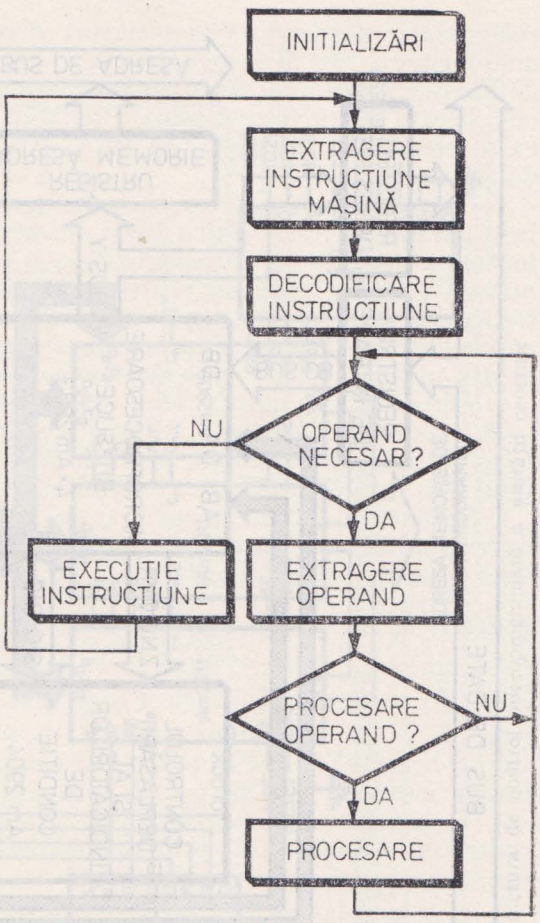


Fig. 10.17. Organigrama generală de control pentru o unitate centrală de calculator

Microinstrucțiunea de control este de tip orizontal și are formatul general din figura 10.19, corespunzător principalelor resurse hardware ale mașinii.

În paragrafele următoare vom detalia structura hardware a unității centrale, microinstrucțiunea de control. Vom prezenta, de asemenea, cîteva din algoritmele de bază implementate prin microcod de o astfel de mașină microprogramată.

10.2.2. STRUCTURA DE CONTROL MICROPROGRAMATĂ

Controlul microprogramat al unității centrale este asigurat de o structură de control microprogramată, SCM, a cărei schemă este dată în figura 10.20.

SCM din figura 10.20 conține un registru pentru încărcarea codului instrucțiune, op-codului, de pe *bus*-ul de date. Este vorba despre porțiunea de

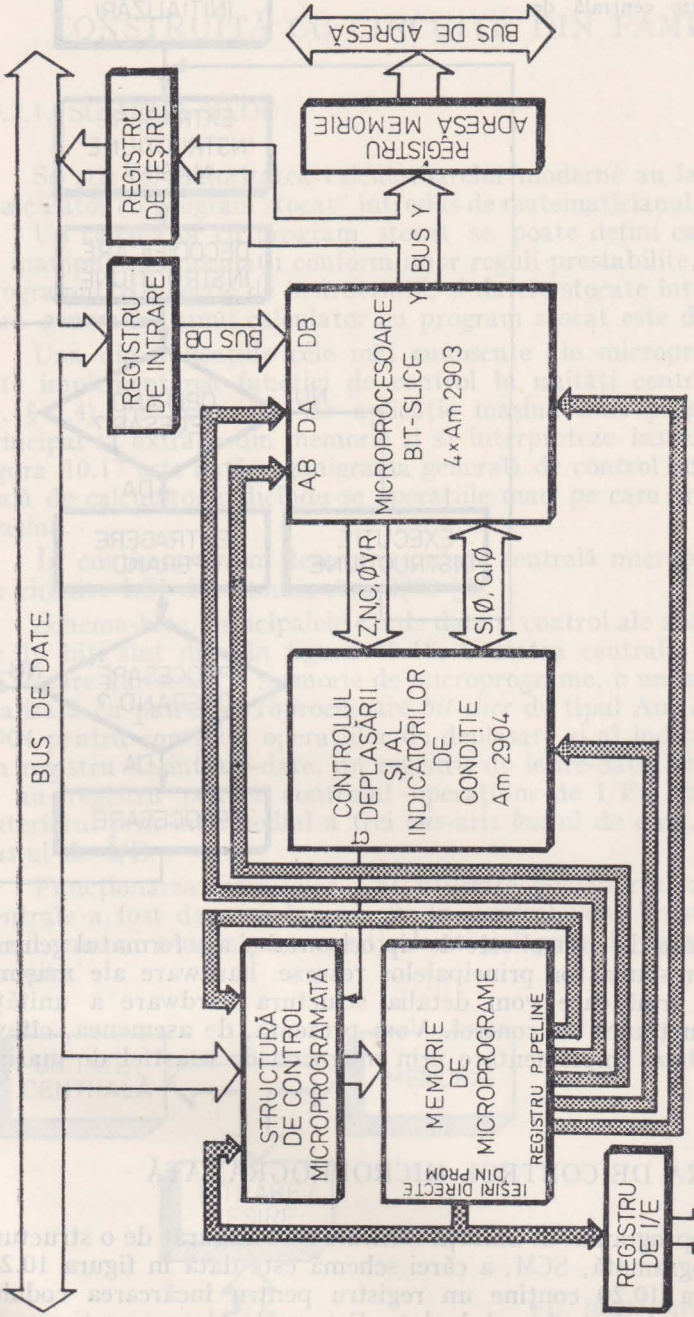


Fig. 10.18. Schema-bloc a unei unități centrale pe 16 biți

Fig. 10.19. Formatul general al microinstrucțiunii

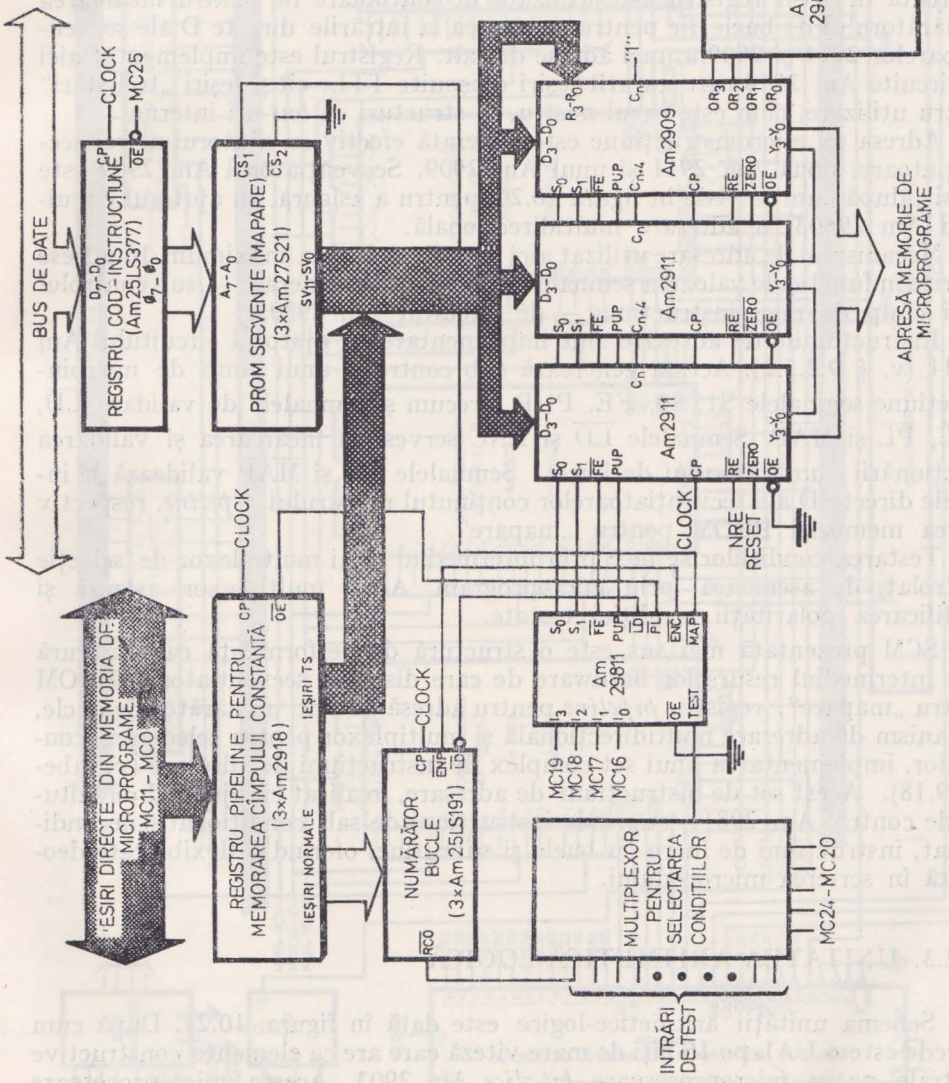


Fig. 10.20. Structura de control microprogramată a unității centrale

cod-operație din instrucțiunea-mașină decodificată mai departe cu ajutorul unei memorii PROM pentru „mapare". Această memorie generează adresele de început ale secvențelor de microinstrucțiuni ce interpretează în mod specific fiecare instrucțiune-mașină. PROM-ul pentru „mapare" poate fi înlocuit cu o rețea logică integrată, un PLA, care generează combinațional adresele de secvențe.

Registrul *pipeline* servește la memorarea unui cîmp de constantă din microinstrucțiune, cîmp ce este citit direct din memoria de microprograme. Constanta încărcată în acest registru este utilizată în continuare fie pentru încărcarea numărătorului de bucle, fie pentru generarea la intrările directe D ale secvențiatoarelor 2911 și 2909 a unei adrese de salt. Registrul este implementat aici cu circuite Am 2918 care au atît ieșiri obișnuite TTL, cît și ieșiri „trei-stări" pentru utilizare, cum este cazul nostru, în structuri cu *bus*-uri interne.

Adresa de microinstrucțiune este generată efectiv cu ajutorul a trei secvențiatoare, două Am 2911 și unul Am 2909. Secvențiatorul Am 2909 este folosit, după cum se vede în figura 10.20, pentru a asigura, cu ajutorul circuitului Am 29803, o adresare multidirecțională.

Mecanismul de adresare utilizat aici permite saltul la maximum 16 adrese diferite în funcție de valoarea semnalelor OR3—OR0 generate — sub controlul unui cîmp de microinstrucțiune — de circuitul Am 29803.

Instrucțiunile de adresare sînt implementate cu ajutorul circuitului Am 29811 (v. § 9.2.5.2). Acesta generează sub controlul unui cîmp de microinstrucțiune semnalele S1, S0, $\overline{FE}$, $\overline{PUP}$, precum și semnalele de validare $\overline{LD}$, $\overline{ENC}$, $\overline{PL}$ și $\overline{MAP}$. Semnalele $\overline{LD}$ și $\overline{ENC}$ servesc la încărcarea și validarea funcționării numărătorului de bucle. Semnalele $\overline{PL}$ și $\overline{MAP}$ validează la intrările directe D ale secvențiatoarelor conținutul registrului *pipeline*, respectiv ieșirea memoriei PROM pentru „mapare".

Testarea condițiilor se face prin intermediul unui multiplexor de selecție controlat, de asemenea, prin microprogram. Acest multiplexor asigură și modificarea polarității condiției testate.

SCM prezentată mai sus este o structură de performanță care asigură prin intermediul resurselor hardware de care dispune, secvențiatoare, PROM pentru „mapare", registru *pipeline* pentru adresă de salt, numărător de bucle, mecanism de adresare multidirecțională și multiplexor pentru selectarea condițiilor, implementarea unui set complex de instrucțiuni de adresare (v. tabelul 9.18). Acest set de instrucțiuni de adresare, realizat cu ajutorul circuitului de control Am 29811, cuprinde instrucțiuni de salt condiționat și necondiționat, instrucțiuni de lucru cu bucle și subrutine, oferind o flexibilitate deosebită în scrierea microcodului.

10.2.3. UNITATEA ARITMETICĂ-LOGICĂ

Schema unității aritmetice-logice este dată în figura 10.21. După cum se vede este o UAL pe 16 biți de mare viteză care are ca elemente constructive centrale patru microprocesoare *bit-slice* Am 2903. Aceste microprocesoare sînt conectate în cascadă într-o structură cu transport anticipat generat de

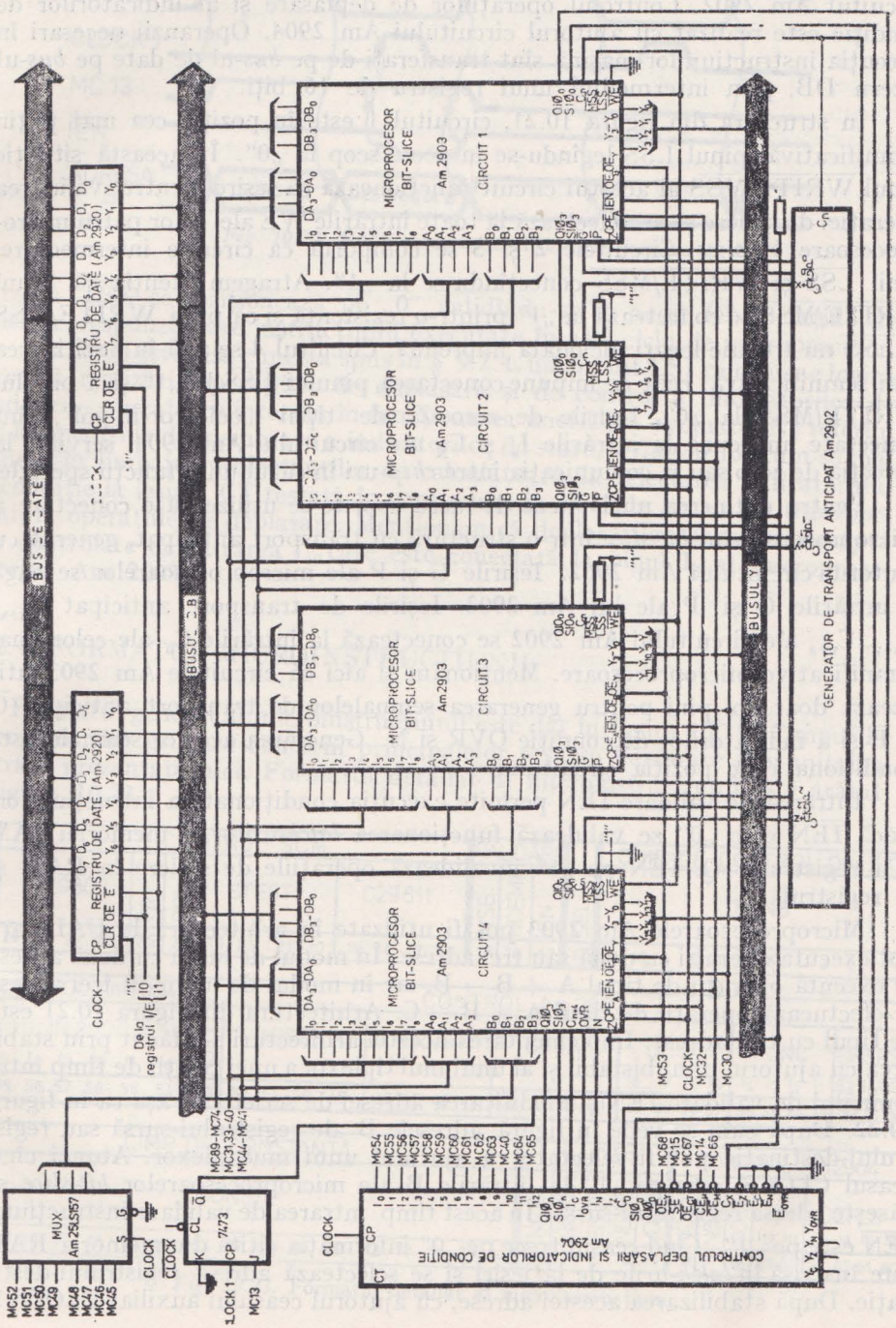


Fig. 10.21. Unitatea aritmetică-logică

circuitul Am 2902. Controlul operațiilor de deplasare și al indicatorilor de condiție este realizat cu ajutorul circuitului Am 2904. Operanzii necesari în execuția instrucțiunilor-mașină sînt transferați de pe *bus*-ul de date pe *bus*-ul intern DB, prin intermediul unui registru de 16 biți.

În structura din figura 10.21, circuitul 1 este în poziția cea mai puțin semnificativă, pinul $\overline{LSS}$ legîndu-se în acest scop la „0". În această situație pinul $\overline{WRITE/MSS}$ al acestui circuit funcționează ca ieșire pentru validarea operației de scriere și se conectează la toate intrările $\overline{WE}$ ale celor patru microprocesoare *bit-slice*. Circuitele 2 și 3 se comportă ca circuite intermediare, pinii $\overline{LSS}$ și $\overline{WRITE/MSS}$ conectîndu-se la „1". Atragem atenția că pinul $\overline{WRITE/MSS}$ se conectează la „1" printr-o rezistență și că pinii $\overline{WRITE/MSS}$ și $\overline{LSS}$ nu trebuie legați niciodată împreună. Circuitul 4 se află în poziția cea mai semnificativă, ceea ce impune conectarea pinului $\overline{LSS}$ la „1" și a pinului $\overline{WRITE/MSS}$ la „0". Ieșirile de zero Z, de tipul „colector-în-gol", sînt conectate împreună la intrările I_z și C_X ale circuitului Am 2904, servind la detecția de zero sau la comunicația între *chip*-uri în cazul unor funcții speciale.

Pentru obținerea unei viteze de lucru sporite se utilizează o conectare a microprocesoarelor *bit-slice* într-o structură cu transport anticipat, generat cu ajutorul circuitului Am 2902. Ieșirile $\overline{G}$ și $\overline{P}$ ale microprocesoarelor se leagă la intrările $\overline{G}$ și $\overline{P}$ ale lui Am 2902. Ieșirile de transport anticipat C_{n+x}, C_{n+y}, C_{n+z} ale circuitului Am 2902 se conectează la intrările C_n ale celor mai semnificative microprocesoare. Menționăm și aici că circuitele Am 2903 utilizează doar doi pini pentru generarea semnalelor de transport anticipat $\overline{G}$ și $\overline{P}$ și a indicatorilor de condiție OVR și N. Generarea acestor semnale este condiționată de poziția circuitului.

Intrarea de validare $\overline{IEN}$ permite execuția condiționată a intrucțiunilor. Dacă $\overline{IEN}$ este „0" se validează funcționarea *latch*-urilor, a memoriei RAM și a registrului Q. $\overline{IEN}$ pe „1" invalidează operațiile de scriere în RAM și în registrul Q.

Microprocesoarele Am 2903 pot fi utilizate în arhitecturi de UAL care pot executa operații cu două sau trei adrese. În modul de lucru cu două adrese se execută operații de tipul A + B → B, iar în modul de lucru cu trei adrese se efectuează operații de tipul A + B → C. Arhitectura din figura 10.21 este de tipul cu trei adrese. Implementarea acestei arhitecturi s-a făcut prin stabilirea cu ajutorul unui bistabil și al unui multiplexor a unei relații de timp între semnalul de validare $\overline{IEN}$ și modificarea adresei de scriere B, așa ca în figura 10.22. După cum se vede în figură, adresele B ale registrului-sursă sau registrului-destinație pot fi selectate cu ajutorul unui multiplexor. Atunci cînd ceasul CLOCK este pe „1" la intrările B ale microprocesoarelor *bit-slice* se găsește adresa registrului-sursă. În acest timp intrarea de validare-instrucțiune $\overline{IEN}$ este pe „1". Cînd ceasul trece pe „0" informația citită din memoria RAM este înscrisă în *latch*-urile de la ieșiri și se selectează adresa registrului-destinație. După stabilizarea acestei adrese, cu ajutorul ceasului auxiliar CLOCK1,

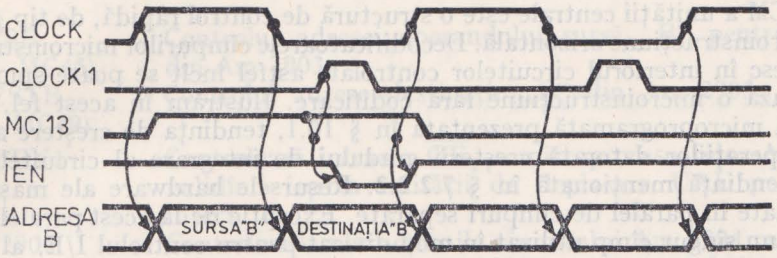


Fig. 10.22. Modificarea adresei de citire/scriere

intrarea $\overline{IEN}$ se poziționează pe „0" validîndu-se în acest fel operațiile de scriere corespunzătoare instrucțiunii executate în acel ciclu de microprocesor.

Am 2904 este, așa cum s-a spus în § 9.2.4, un circuit LSI ce conține logica necesară realizării operațiilor de deplasare și de control al indicatorilor de condiție — operații ce intervin în funcționarea unei UAL evoluate. Ne referim aici la operațiile de memorare a indicatorilor de stare, a *flag*-urilor cum se mai numesc în limbajul proiectanților de calculatoare, la selectarea indicatorului de condiție la ieșirea CT testată în continuare în SCM, la generarea transportului, la operațiile de deplasare. Menționăm că deplasările stînga sau dreapta sînt controlate de intrarea I_{10} care este conectată la pinul I_8 al microprocesoarelor Am 2903.

10.2.4. FORMATUL MICROINSTRUCȚIUNII

Formatul general al microinstrucțiunii este dat în figura 10.19. Cîmpurile de control sînt grupate conform principalelor resurse hardware din unitatea centrală prezentată aici. Formatul detaliat al microinstrucțiunii este arătat în figura 10.23.

	CONTROL SCM									VALIDĂRI				CÎMP DE CONTROL DIVIZAT										
	C29803			VRI	POL	STEST			C29811	NSE	NCEMI	IEN	SEL	IESNB										
MC	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

	CONTROL Am2904							CONTROL Am2903																
	IN 2904						NOEY1	DESTB				SRSB				SRSA				DEST				FUNC				OPSRS			
	I_5	I_4	I_3	I_2	I_1	I_0														I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	$\overline{E}_A$	I_0	$\overline{OE}_B$
MC	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30

	CÎMP DE CONSTANTĂ																VALIDĂRI				CONTROL Am2904							
	CONST																NRE	NCESR	IENSR	NCZER	NCONST	NOEY2	NOECT	NCEM	CARRY		DEPLS		
																									I_{12}	I_{11}	I_9	I_8	I_7	I_6
MC	99	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60

Fig. 10.23. Formatul detaliat al microinstrucțiunii

SCM a unității centrale este o structură de control rapidă, de tip *pipeline*, cu microinstrucțiune orizontală. Decodificatoarele cîmpurilor microinstrucțiunii se găsesc în interiorul circuitelor controlate astfel încît se poate spune că se utilizează o microinstrucțiune fără codificare. Ilustrăm în acest fel, față de mașina microprogramată prezentată în § 10.1, tendința de creștere a puterii microoperațiilor datorată creșterii gradului de integrare al circuitelor utilizate, tendință menționată în § 7.2.2.2. Resursele hardware ale mașinii sînt controlate în paralel de cîmpuri separate. Excepție de la acest paralelism face numai un singur cîmp utilizat în mod divizat pentru controlul I/E, al saltului în microprogram sau pentru încărcarea numărătorului de bucle din SCM.

În continuare vom prezenta succint funcțiile cîmpurilor microinstrucțiunii de control a unității centrale pe care o discutăm aici:

IESNB (MC11—MC0)	Cîmp de control divizat utilizat pentru controlul I/E, al salturilor în microprogram sau pentru încărcarea numărătorului de bucle din SCM.
SEL (MC12)	Validează încărcarea cîmpului IESNB în registrul de control I/E.
IEN (MC13)	Validează atunci cînd este pe „1" ieșirea $\overline{WRITE}$ permițînd înscrierea registrului Q și a bistabilului de comparare-semn din Am 2903 (fig. 10.22).
NCEMI (MC14)	Controlează intrarea $\overline{CE}_\mu$ din Am 2904 care validează încărcarea registrului μSR.
NSE (MC15)	Controlează intrarea $\overline{SE}$ din Am 2904 care validează operațiile de deplasare.
C29811 (MC19—MC16)	Cîmp pentru controlul circuitului Am 29811 din SCM. Acest cîmp stabilește instrucțiunea de adresare conform tabelului 9.18.
STEST (MC23—MC20)	Cîmp care controlează multiplexorul pentru selecția condiției testate în SCM.
PØL (MC24)	Stabilește polaritarea condiției testate.
VRI (MC25)	Validează încărcarea registrului-instrucțiune din SCM.
C29803 (MC29—MC26)	Cîmp pentru controlul circuitului Am 29803.
ØPSRS (MC32—MC30)	Selectează operanzii sursă pentru UAL din Am 2903, conform tabelului 9.6, reprezentînd de fapt intrările $\overline{E}_A$, I_0 și $\overline{ØE}_B$.
FUNC (MC36—MC33)	Selectează funcțiunile microprocesorului Am 2903 conform tabelului 9.7, reprezentînd intrările I_4—I_1.
DEST (MC40—MC37)	Selectează destinația UAL stabilind și funcțiile speciale ale microprocesorului Am 2903, conform tabelelor 9.8 și 9.9.
SRSA (MC44—MC41)	Controlul adresei operandului sursă „A" pentru UAL din Am 2903.

SRSB (MC48—MC45)	Controlul adresei operandului sursă „B" pentru UAL din Am 2903.
DESTB (MC52—MC49)	Controlul adresei-destinație „B" din Am 2903.
NØEY1 (MC53)	Controlează intrarea $\overline{OE}_Y$ a microprocesoarelor Am 2903 validînd ieșirea unității de deplasare UAL pe *bus*-ul intern Y.
IN2904 (MC59—MC54)	Controlează instrucțiunile circuitului Am 2904 care folosesc registrele μSR și MSR, conform tabelelor 9.10, 9.11 și 9.13.
DEPLS (MC63—MC60)	Cîmp de control al multiplexorului pentru deplasări din Am 2904 conform tabelului 9.14.
CARRY (MC65—MC64)	Controlează multiplexorul utilizat pentru generarea transportului în Am 2904, conform tabelului 9.15.
NCEM (MC66)	Controlează intrarea de validare $\overline{CE}_M$ din Am 2904.
NØECT (MC67)	Controlează intrarea de validare $\overline{OE}_{CT}$ din Am 2904.
NØEY2 (MC68)	Controlează intrarea de validare $\overline{OE}_Y$ din Am 2904.
VCØNST (MC69)	Validează cîmpul de constantă MC89—MC74 pe *bus*-ul intern DA al circuitelor Am 2903.
DIV (MC72—MC70)	Diverse semnale de ștergere și validare.
NRE (MC73)	Validează înscrierea constantei IESNB în registrul AR din secvențiatoarele Am 2909 și Am 2911.
CØNST (MC89—MC74)	Cîmp de constantă — 16 biți.

10.2.5. DEFINIREA MICROINSTRUCȚIUNII CU AJUTORUL META-ASAMBLORULUI METASM

În acest paragraf vom defini cu ajutorul meta-asamblorului METASM, prezentat în cap. 8, microinstrucțiunea de control a unității centrale pe care o discutăm aici. De asemenea, vom defini microoperațiile principale, construind deci un limbaj în care se vor scrie cîteva din microrutinele mai des utilizate în proiectarea unităților centrale microprogramate.

```
;
;MICRØPRØGRAM DE CØNTRØL PENTRU UC
;
DEFIN WIDTH 90; LUNGIMEA MICRØINSTRUCȚIUNII
;
;DEFINIREA CÎMPURILØR MICRØINSTRUCȚIUNII
;
```

```
;CÎMP DE CØNSTANTĂ
;
FIELD CØNST WIDTH 16 DEFAULT 0;
;
;VALIDARI
;
FIELD NRE WIDTH 1 DEFAULT 1; VALIDARE INSCRIERE AR DIN
; AM2909,AM2911
FIELD DIV WIDTH 3 DEFAULT 101B; DIVERSE
FIELD VCØNST WIDTH 1 DEFAULT 1; VALIDARE CØNSTANTA
; PE BUSUL DA
;
;CØNTRØL AM2904
;
FIELD NØEY2 WIDTH 1 DEFAULT 1; INTRAREA NØEY
FIELD NØECT WIDTH 1 DEFAULT 1; INTRAREA NØECT
FIELD NCEM WIDTH 1 DEFAULT 1; INTRAREA NCEM
FIELD CARRY WIDTH 2 DEFAULT 0; CØNTRØLUL TRANSPØRTU-
;LUI
FIELD DEPLS WIDTH 4 DEFAULT 0; CØNTRØLUL DEPLASĂRILOR
FIELD IN2904 WIDTH 6 DEFAULT 0; INSTRUCȚIUNILE AM2904
;
;CØNTRØL AM2903
;
FIELD NØEY1 WIDTH 1 DEFAULT 1; INTRAREA NØEY
FIELD DESTB WIDTH 4 DEFAULT 0; ADRESA REGISTRU DESTINA-
;TIE „B"
FIELD SRSB WIDTH 4 DEFAULT 0; ADRESA REGISTRU SURSA „B"
FIELD SRSA WIDTH 4 DEFAULT 0; ADRESA REGISTRU SURSA „A"
FIELD DEST WIDTH 4 DEFAULT 0; DESTINATIE UAL
FIELD FUNC WIDTH 4 DEFAULT 0; FUNCTIA UAL
FIELD ØPSRS WIDTH 3 DEFAULT 0; ØPERANZII SURSA UAL
;
;CØNTRØL SCM
;
FIELD C29803 WIDTH 4 DEFAULT 0; CØNTRØL AM29803 (SALT
;MULTIDIRECTIONAL)
FIELD VRI WIDTH 1 DEFAULT 1; VALIDARE REGISTRU INSTRUC-
;TIUNE
FIELD PØL WIDTH 1 DEFAULT 0; PØLARITATE TEST
FIELD STEST WIDTH 4 DEFAULT 0; SELECTIE CØNDITIE TESTATA
FIELD C29811 WIDTH 4 DEFAULT 0; CØNTRØL AM29811 (INSTR.
;ADRESARE)
;
;VALIDARI
;
```

```
FIELD NSE WIDTH 1 DEFAULT 1; INTRAREA NSE A CIRCUITULUI
;AM2904
FIELD NCEMI WIDTH 1 DEFAULT 0; VALIDARE MICRØ STATUS
;REGISTER
FIELD IEN WIDTH 1 DEFAULT 0; VALIDARE SCRIERE AM2903
FIELD SEL WIDTH 1 DEFAULT 1; SELECTEAZĂ FUNCTIA CIMPU-
;LUI IESNB
FIELD IESNB WIDTH 12 DEFAULT 0; CÎMP DIVIZAT
;I/E, SALT, NUM. BUCLE
;
ENDDEF; SFIRSIT DEFINIRE MICRØINSTRUCTIUNE
;
;DEFINIRE MICRRØØPERATII
;
;DESTINATIE UAL (V. TAB. 9.9)
;
ADR:EQU 0H;DEPL.ARITM. DR., REZULTATUL IN RAM
LDR:EQU 1H; DEPL. LØG. DR.
ADRQ:EQU 2H; DEPL. ARITM. DR., REZULTATUL IN RAM SI Q
LDRQ:EQU 3H; DEPL. LØG. DR., REZULTATUL IN RAM SI Q
RPT:EQU 4H; REZULTATUL IN RAM, GENERARE PARITATE
LDQP:EQU 5H; DEPL. LØG. DR. A REG. Q, GENERARE PARITATE
QPT:EQU 6H; REZULTATUL IN REG. Q, GENARARE PARITATE
RQPT: EQU 7H; REZULTATUL IN RAM SI Q, GENERARE PARITATE
AUR:EQU 8H; DEPL. ARITM. ST., REZULTATUL IN RAM
LUR:EQU 9H; DEPL. LØG. ST., REZULTATUL IN RAM
AURQ:EQU 0AH; DEPL. ARITM. ST., REZULTATUL IN RAM SI Q
LURQ:EQU 0BH;DEPL. LØG. ST., REZULTATUL IN RAM SI Q
YBUS:EQU 0CH; REZULTATUL NUMAI PE BUSUL Y
LUQ:EQU 0DH;DEPL. LØG. ST., A REG. Q
SINX:EQU 0EH;EXTENSIE DE SEMN
REG:EQU 0FH;REZULTATUL IN RAM, EXTENSIE DE SEMN
;
;ØPERANZII SURSA UAL (V. TAB. 9.6)
;
RADB:EQU 001B;RAM „A", DB
RAQ:EQU 010B;RAM „A", Q
DARB:EQU 100B;DA, RAM „B"
DADB:EQU 101B;DA,DB
DAQ:EQU 110B;DA, Q
;
;REGISTRE DE LUCRU RAM
;
R0:EQU 0H;
R1:EQU 1H;
R2:EQU 2H;
R3:EQU 3H;
R4:EQU 4H;
R5:EQU 5H;
```

```
R6:  EQU 6H;
R7:  EQU 7H;
R8:  EQU 8H;
R9:  EQU 9H;
R10: EQU 0AH;
R11: EQU 0BH;
R12: EQU 0CH;
R13: EQU 0DH;
R14: EQU 0EH;
R15: EQU 0FH;
;
;CØNTRØLUL TRANSPØRTULUI (V. TAB. 9.15)
;
ØNE:EQU 01B;CARRY—IN = 1
CZ:EQU 10B;CARRY—IN=Z
;
;MICRØØPERATII SCM
;
;SELECTIE CØNDITII
;
MICRØP GRD ASSIGN STEST=0;GRØUND
MICRØP CNT ASSIGN STEST=0FH;CØUNTER
;
;INSTRUCTIUNI DE ADRESARE AM29811 (V. TAB. 9.18)
;
MICRØP JZ ASSIGN C29811=0H;SALT LA ADRESA ZERØ
MICRØP CJS ASSIGN C29811=1H;SALT CØND. LA SUBRUTINA/
;REG. PIPELINE
MICRØP JMAP ASSIGN C29811=2H;SALT MAPARE
MICRØP CJP ASSIGN C29811=3H;SALT CØND.-REG. PIPELINE
MICRØP PUSH ASSIGN C29811=4H;PUSH/INCARCARE CØND. NUM.
;BUCLE
MICRØP JSRP ASSIGN C29811=5H;SALT LA SUBRUTINA AR/REG.
;PIPELINE
MICRØP CJV ASSIGN C29811=6H;SALT CØND.-VECTOR
MICRØP JRP ASSIGN C29811=7H;SALT CØND. LA AR/REG. PIPE-
;LINE
MICRØP RFCT ASSIGN C29811=8H;REPETA BUCLA DE LA ADRESA
;DIN STIVA
MICRØP RPCT ASSIGN C29811=9H;REPETA BUCLA DE LA ADRESA
;DIN REG. PIPE.
MICRØP CRTN ASSIGN C29811=0AH;REVENIRE CØND. DIN
;SUBRUTINA
MICRØP CJPP ASSIGN C29811=0BH;SALT CØND. LA REG. PIPELINE/
;STIVA
MICRØP LDCT ASSIGN C29811=0CH; INCARCARE REGISTRU SI
;„CØNTINUE"
MICRØP LØØP ASSIGN C29811=0DH; TEST SFIRSIT BUCLA
```

```
MICRØP CØNT ASSIGN C29811=0EH; „CØNTINUE"
MICRØP JP ASSIGN C29811=0FH;SALT LA ADRESA DIN REG. PI-
;PELINE
MICRØP JSR ASSIGN GRD F C29811=1H;SALT LA SUBRUTINA AR
MICRØP RTN ASSIGN GRD F C29811=0AH;REVENIRE DIN SUBRU-
;TINA
;
;CIMPUL DE CØNTRØL DIVIZAT
;
MICRØP GØTØ ADRESA ASSIGN SEL=1 IESNB=ADRESA; ADRESA
;DE SALT
MICRØP CØUNT NUMAR ASSIGN SEL=1 IESNB=NUMAR; NUMARUL
;DE BUCLE
MICRØP PUT CØNST ASSIGN SEL=0 IESNB=CØNST; ØPERATIE
;I/E
;
;PØLARITATEA TESTULUI
;
MICRØP T ASSIGN PØL=1; VALØAREA ADEVARATA
MICRØP F ASSIGN PØL=0;VALØAREA NEGATA
;
;FUNCTII AM2903 (V. TAB. 9.7).
;
MICRØP IN ASSIGN NØEY1=1 DESTB=R15 DEST=0FH
FUNC=0H IEN=1;
MICRØP ØUT X ASSIGN NØEY1=0 SRSA=R15 DEST=0CH
FUNC=6H ØPSRS=X IEN=1;
MICRØP YØFF ASSIGN NØEY=1;
MICRØP HIGH X,W ASSIGN NØEY1=0 DESTB=X DEST=W
FUNC=0 ØPSRS=010B;
MICRØP SRS X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=1 ØPSRS=XX IEN=1;
MICRØP SSR X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=2 ØPSRS=XX IEN=1;
MICRØP ADD X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=3 ØPSRS=XX IEN=1;
MICRØP PAS X,Y,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
DEST=W FUNC=4 ØPSRS=XX IEN=1;
MICRØP CØMS X,Y,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
DEST=W FUNC=5 ØPSRS=XX IEN=1;
MICRØP PAR X,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSA=Z
DEST=W FUNC=6 ØPSRS=XX IEN=1;
MICRØP CØMR X,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSA=Z
DEST=W FUNC=7 ØPSRS=XX IEN=1;
MICRØP LØW X,W ASSIGN NØEY1=0 DESTB=X DEST=W
FUNC=8 IEN=1;
MICRØP CRAS X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=9 ØPSRS=XX IEN=1;
```

```
MICRØP XNRS X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=0AH ØPSRS=XX IEN=1;
MICRØP XØR X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=0BH ØPSRS=XX IEN=1;
MICRØP AND X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=0CH ØPSRS=XX IEN=1;
MICRØP NØR X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=0DH ØPSRS=XX IEN=1;
MICRØP NAND X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=0EH ØPSRS=XX IEN=1;
MICRØP ØR X,Y,Z,W,XX ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=W FUNC=0FH ØPSRS=XX IEN=1;
;
;FUNCTII SPECIALE AM2903 (V. TAB. 9.8), ALTE FUNCTII
;
MICRØP UMUL X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=0 FUNC=0 ØPSRS=0;
MICRØP TCM X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y SRSA=Z
DEST=2 FUNC=0 ØPSRS=0;
MICRØP INC X,Y ASSIGN NØEY1=0 DESTB=X SRSB=Y DEST=4
FUNC=0 ØPSRS=0;
MICRØP SMTC X,Y ASSIGN NØEY1=0 DESTB=X SRSB=Y DEST=5
FUNC=0 ØPSRS=0;
MICRØP TCMC X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=6 FUNC=0 ØPSRS=0;
MICRØP SLN X,Y ASSIGN NØEY1=0 DESTB=X SRSB=Y DEST=8
FUNC=0 ØPSRS=0;
MICRØP DLN X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y SRSA=Z
DEST=0AH FUNC=0 ØPSRS=0;
MICRØP TDIV X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y
SRSA=Z DEST=0CH FUNC=0 ØPSRS=0;
MICRØP TDC X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y SRSA=Z
DEST=0EH FUNC=0 ØPSRS=0;
MICRØP SDQP ZZ ASSIGN NØEY1=0 DEST=5 ØPSRS=ZZ IEN=1;
MICRØP SUQP ZZ ASSIGN NØEY1=0 DEST=0DH ØPSRS=ZZ IEN=1;
MICRØP LQPT Z,ZZ ASSIGN NØEY1=0 SRSA=Z DEST=6 FUNC=6
ØPSRS=ZZ IEN=1;
MICRØP RMØV X,Y,Z ASSIGN NØEY1=0 DESTB=X SRSB=Y
DEST=Z FUNC=4 ØPSRS=ZZ IEN=1;
MICRØP QMØV X ASSIGN NØEY1=0 DESTB=X DEST=0FH
FUNC=4 ØPSRS=010B IEN=1;
MICRØP SDRL X,Y,ZZ ASSIGN NØEY1=0 DESTB=X SRSB=Y
DEST=1 FUNC=4 ØPSRS=ZZ IEN=1;
MICRØP SURL X,Y,ZZ ASSIGN NØEY1=0 DESTB=X SRSB=Y
DEST=9 FUNC=4 ØPSRS=ZZ IEN=1;
MICRØP SDRQ X,Y,ZZ ASSIGN NØEY1=0 DESTB=X SRSB=Y
DEST=3 FUNC=4 ØPSRS=ZZ IEN=1;
;
```

```
;CØNTRØLUL CIRCUITULUI AM2904
;
;ØPERATII DE DEPLASARE (V. TAB. 9.14)
;
MICRØP SDL   ASSIGN NSE=0  DEPLS=0;I10=0(I8—AM2903)
MICRØP SDH   ASSIGN NSE=0  DEPLS=1;I10=0(I8—AM2903)
MICRØP SUL   ASSIGN NSE=0 DEPLS=2;I10=1(I8—AM2903)
MICRØP SUH   ASSIGN NSE=0 DEPLS=3;I10=1(I8—AM2903)
MICRØP SDDH ASSIGN NSE=0 DEPLS=3; I10=0(I8—AM2903)
MICRØP SDDL ASSIGN NSE=0 DEPLS=6;I10=0(I8—AM2903)
MICRØP SDUL ASSIGN NSE=0 DEPLS=6; I10=1(I8—AM2903)
MICRØP SDUH ASSIGN NSE=0 DEPLS=7; I10=1(I8—AM2903)
MICRØP RSD   ASSIGN NSE=0 DEPLS=0AH;I10=0(I8—AM2903)
MICRØP RSU   ASSIGN NSE=0 DEPLS=0AH;I10=1(I8—AM2903)
MICRØP SSXØ ASSIGN NSE=0 DEPLS=0EH;I10=0(I8—AM2903)
MICRØP RDD   ASSIGN NSE=0 DEPLS=0FH;I10=0(I8—AM2903)
MICRØP RDU   ASSIGN NSE=0 DEPLS=0FH;I10=1(I8—AM2903)
MICRØP SDMS ASSIGN NSE=0 DEPLS=5; I10=0(I8—AM2903)
MICRØP SMS   ASSIGN NSE=0 DEPLS=2;I10=0(I8—AM2903)
MICRØP SDDC ASSIGN NSE=0 DEPLS=7; I10=0(I8—AM2903)
MICRØP SDUC ASSIGN NSE=0 DEPLS=4;I10=1(I8—AM2903)
;
;ØPERATII CU MICRØ STATUS REGISTER (V. TAB. 9.10)
;
MICRØP RSTI ASSIGN IN2904=03Q;
MICRØP SWAP ASSIGN IN2904=02Q;
SHLD:EQU 1B;
;
;ØPERATII CU MACHINE STATUS REGISTER (V. TAB. 9.11)
;
MICRØP LMA ASSIGN NCEM=0 IN2904=00Q;
MICRØP RSTA ASSIGN NCEM=0 IN2904=03Q;
MICRØP SHØLD ASSIGN NCEM=0;
;
;SELECTIA IESIRII DE TEST CT (V. TAB. 9.13)
;
MICRØP MIZ   ASSIGN NØECT=0 IN2904=24Q STEST=0BH;
MICRØP MIØ   ASSIGN NØECT=0 IN2904=26Q STEST=0BH;
MICRØP MIC   ASSIGN NØECT=0 IN2904=32Q STEST=0BH;
MICRØP MIS   ASSIGN NØECT=0 IN2904=36Q STEST=0BH;
MICRØP MAZ   ASSIGN NØECT=0 IN2940=44Q STEST=0BH;
MICRØP MAØ   ASSIGN NØECT=0 IN2904=46Q STEST=0BH;
MICRØP MAC   ASSIGN NØECT=0 IN2904=52Q STEST=0BH;
MICRØP MAS   ASSIGN NØECT=0 IN2904=56Q STEST=0BH;
;
;INVALIDARI
;
MICRØP ALUØFF ASSIGN IEN=0;
```

```
MICRØP ALLØFF ASSIGN NSE=1 NCEMI=1 IEN=0;
;
;INCARCARE CØNSTANTA
;
MICRØP CØNST XXX ASSIGN CØNST=XXX VCØNST=0;
;
;
;MICRØPRØGRAMUL DE CØNTRØL
;
;
MPRØG MPUC WIDTH 90;
.....
```

10.2.6. CÎTEVA MICRORUTINE MAI DES UTILIZATE

10.2.6.1. **Microrutina de înmulțire a două numere binare fără semn**

Algoritmul folosit are la bază o metodă clasică de înmulțire directă a unor numere reprezentate în mărime și semn [4].

Deînmulțitul X și înmulțitorul Y, numere de 16 biți, se reprezintă astfel:

$$X = \sum_{i=0}^{15} x_i \cdot 2^i; \tag{10.7}$$

$$Y = \sum_{i=0}^{15} y_i \cdot 2^i, \tag{10.8}$$

unde x_i, $y_i \in \{0, 1\}$.

Produsul U va fi dat de:

$$U = X \cdot Y = X \cdot y_{15} \cdot 2^{15} + \ldots + X \cdot y_1 \cdot 2^1 + X \cdot y_0 \cdot 2^0. \tag{10.9}$$

Deoarece coeficienții y sînt 0 sau 1 înmulțirea binară este de fapt, conform relației (10.9), o adunare repetată a deînmulțitului deplasat corespunzător.

În algoritmul utilizat aici se formează la fiecare adunare cîte un așa-numit produs parțial. Primul produs parțial este:

$$p_1 = X \cdot y_0 \tag{10.10}$$

Al doilea produs parțial p_2 este:

$$p_2 = p_1 \cdot 2^{-1} + X \cdot y_1. \tag{10.11}$$

Al 16-lea produs parțial p_{16} va fi dat de:

$$p_{16} = p_{15} \cdot 2^{-1} + X \cdot y_{15} \tag{10.12}$$

Înlocuind toate produsele parțiale în (10.12) obținem:

$$p_{16} = X \cdot (y_{15} + \ldots + y_1 \cdot 2^{-14} + y_0 \cdot 2^{-15}) \tag{10.13}$$

Înmulțind în ambele părți cu 2^{15} se obține:

$$p_{16} \cdot 2^{15} = X \cdot (y_{15} \cdot 2^{15} + \ldots + y_1 \cdot 2^1 + y_0 \cdot 2^0) = X \cdot Y \quad (10.14)$$

Produsul U este dat deci de:

$$U = p_{16} \cdot 2^{15}, \quad (10.15)$$

iar produsele parțiale de expresia:

$$p_i = p_{i-1} \cdot 2^{-1} + X \cdot y_{i-1}. \quad (10.16)$$

Formulele (10.15) și (10.16) reprezintă algoritmul de înmulțire a două numere binare fără semn ce va fi implementat mai jos prin microcod.

Organigrama microrutinei de înmulțire a două numere binare de 16 biți este dată în figura 10.24. Algoritmul realizat cu ajutorul funcției speciale UMUL impune ca inițial, la intrarea în subrutină, locația RAM adresată de cîmpul SRSB să fie zero, înmulțitorul să se afle în registrul Q și deînmulțitul într-o locație adresată de cîmpul SRSA. Deci pentru a intra în microrutina de înmulțire descrisă aici este necesar ca registrul R1 să fie zero, deînmulțitul să se găsească în registrul R0 și înmulțitorul în registrul R15. La sfîrșitul subrutinei rezultatul reprezentat pe 32 biți se va găsi în registrele R1 și Q.

Microcodul care efectuează înmulțirea efectivă este dat mai jos.

```
;INMULȚIRE FARA SEMN
IFS: .....
     LQPT R15,0 GRD F PUSH CØUNT 00FH;
     UMUL R1,R1,R0 IEN=1 SDDL CNT F RFCT;
     .....
```

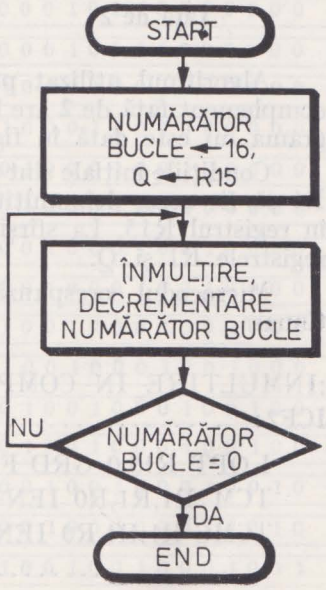


Fig. 10.24. Organigrama microrutinei de înmulțire a două numere binare fără semn

Prima microinstrucțiune transferă registrul R15, care conține înmulțitorul, în registrul Q și pregătește intrarea în bucla de înmulțire. În acest scop se memorează în stivă adresa de început a buclei și în numărătorul de bucle, numărul de buclări. În continuare se intră în buclă executîndu-se de 16 ori funcția specială de înmulțire UMUL.

În timpul microinstrucțiunii de înmulțire pinul Z al celui mai puțin semnificativ microprocesor *bit-slice* este ieșire, ceilalți pini Z fiind intrări. Pinul Z al circuitului 1 are valoarea celui mai puțin semnificativ bit al registrului Q. Această ieșire Z validează deci, atunci cînd este „1", adunarea produsului parțial din registrul R1, adresat de cîmpul SRSB, cu deînmulțitul din registrul R0, adresat de cîmpul SRSA. În timpul înmulțirii C_n este „0".

La fiecare front pozitiv al ceasului ieșirea UAL va fi deplasată dreapta, rezultatul deplasării memorîndu-se în registrul R1 adresat de cîmpul DESTB al microinstrucțiunii. În timpul înmulțirii se formează un registru de 32 biți a cărei deplasare dreapta se execută sub controlul circuitului Am 2904 cu ajutorul microoperației SDDL. La sfîrșitul operației de înmulțire partea cea mai semnificativă a produsului se află în registrul R1 și partea cea mai puțin semnificativă în registrul Q.

Pentru o înțelegere mai ușoară a algoritmului, în figura 10.25 se exemplifică înmulțirea a două numere de 16 biți indicîndu-se operațiile de adunare și deplasare executate cu registrele R1, R0 și Q.

În concluzie, observăm că operația de înmulțire propriu-zisă a două numere binare de 16 biți fără semn durează într-o unitate centrală microprogramată de tipul celei descrise aici 17 microcicli, fiind necesare doar două microinstrucțiuni.

10.2.6.2. Microrutina de înmulțire a două numere reprezentate în complement față de 2

Algoritmul utilizat pentru înmulțirea a două numere reprezentate în complement față de 2 are la bază prima metodă a lui Robertson [4]. Organigrama lui este dată în figura 10.26.

Condițiile inițiale sînt aceleași ca și în cazul înmulțirii fără semn: registrul R1 să fie zero, deînmulțitul să se găsească în registrul R0 și înmulțitorul în registrul R15. La sfîrșitul operației de înmulțire rezultatul se va găsi în registrele R1 și Q.

Microcodul corespunzător organigramei din figura 10.26 este dat în continuare.

```
;INMULTIRE IN COMPLEMENT FATA DE 2
ICF2: .....
    LQPT R15,0 GRD F PUSH CØUNT 00EH;
    TCM R1,R1,R0 IEN=1 SDDL CNT F RFCT;
    TCMC R1,R1,R0 IEN=1 SDDL CARRY=CZ CØNT;
    .....
```

$(R0) = X = 373_{10} = 0175_{16}$ $(Q) = Y = 69_{10} = 0045_{16}$
$(R1, Q) = U = X \cdot Y = 25737_{10} = 00006489_{16}$

	REGISTRUL R1	REGISTRUL Q
		$y_{15}y_{14}y_{13}y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0$
1 +X	0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1
	0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 →	1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
2 +0	0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0	1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
	0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 →	0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
3 +X	0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0	0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
	0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 →	0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
4 +0	0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1	0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
	0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 →	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
5 +0	0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 →	0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
6 +0	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0	0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 →	0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
7 +X	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0	0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
	0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 →	0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
8 +0	0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1	0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 →	1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
9 +0	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0	1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 →	0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
10 +0	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0	0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 →	0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
11 +0	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1	0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 →	1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0
12 +0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0	1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 →	0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
13 +0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 →	0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0
14 +0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 →	1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0
15 +0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 →	1 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0
16 +0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 →	0 1 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0

Fig. 10.25. Exemplu de înmulțire a două numere binare de 16 biți **fără** semn

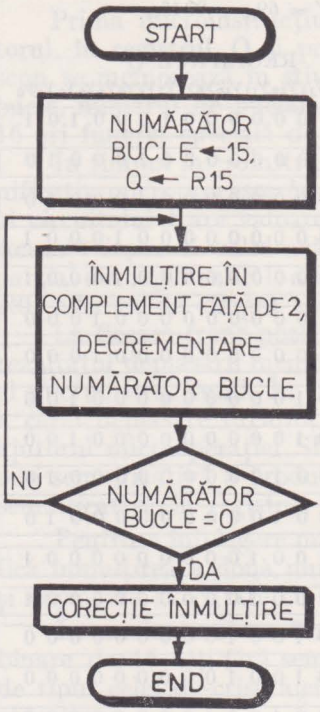


Fig. 10.26. Organigrama microrutinei de înmulțire a două numere reprezentate în complement față de 2

Prima microinstrucțiune transferă registrul R15, care conține înmulțitorul, în registrul Q și pregătește intrarea în bucla de înmulțire memorînd în stivă adresa de început a buclei și numărul de buclări în numărătorul de bucle. Se observă că numărul de bucle este 15 deoarece conform primei metode a lui Robertson ultimul ciclu al înmulțirii reprezintă o corecție a rezultatului.

Urmează bucla de înmulțire propriu-zisă care permite ca funcția specială TCM să fie aplicată de 15 ori. În timpul deplasării dreapta în poziția Y_3 a celui mai semnificativ microprocesor se deplasează semnalul N ∀ OVR generat intern. În aceasta constă și deosebirea între funcția de înmulțire în complement față de 2, TCM, și funcția de înmulțire fără semn, UMUL, în ultimul caz în Y_3, fiind deplasat C_{n+4}.

Ultima microinstrucțiune realizează corecția produsului parțial. Această corecție se efectuează numai în cazul în care înmulțitorul este negativ, Z = 1, prin scăderea deînmulțitului.

Ca și la înmulțirea fără semn, microrutina de înmulțire în complement față de 2 utilizează pentru calculul rezultatului un registru de 32 biți a cărei deplasare se execută sub controlul circuitului Am 2904. La sfîrșitul operației de înmulțire partea cea mai semnificativă a produsului se găsește în registrul R1 din memoria RAM, iar partea mai puțin semnificativă în registrul Q.

Pentru o mai clară înțelegere a algoritmului, în figurile 10.27 și 10.28 se exemplifică înmulțirea a două numere reprezentate în complement față de 2, indicîndu-se operațiile de adunare, scădere și corecție executate cu registrele

$(R0) = X = -13_{10} = FFF3_{16}$ $(Q) = Y = +10_{10} = 000A_{16}$
$(R1, Q) = U = X \cdot Y = -130_{10} = FFFFFF7E_{16}$

	REGISTRUL R1	REGISTRUL Q
		$y_{15}y_{14}y_{13}y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0$
1 +0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 →	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
2 +X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 →	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
3 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 →	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
4 +X	1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
	1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 →	1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
5 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1	1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 →	1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
6 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1	1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 →	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
7 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1	1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 →	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
8 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
9 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0
10 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0
11 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 0
12 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0
13 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0
14 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0
15 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0
16 +0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 →	1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0

Fig. 10.27. Exemplu de înmulțire a două numere de 16 biți reprezentate în complement față de 2 (înmulțitorul pozitiv)

$(R0) = X = -13_{10} = FFF3_{16}$ $(Q) = Y = -10_{10} = FFF6_{16}$

$(R_1, Q) = U = X \cdot Y = 130_{16} = 00000082_{16}$

		REGISTRUL R1	REGISTRUL Q
			$y_{15}y_{14}y_{13}y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0$
1	+0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 →	0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
2	+X	1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1	0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 →	1 0 1 1 1 1 1 1 1 1 1 1 1 0 1
3	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0	1 0 1 1 1 1 1 1 1 1 1 1 1 0 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 →	0 1 0 1 1 1 1 1 1 1 1 1 1 1 0
4	+0	1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0	0 1 0 1 1 1 1 1 1 1 1 1 1 1 0
		1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 →	0 0 1 0 1 1 1 1 1 1 1 1 1 1 1
5	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0	0 0 1 0 1 1 1 1 1 1 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 →	0 0 0 1 0 1 1 1 1 1 1 1 1 1 1
6	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0	0 0 0 1 0 1 1 1 1 1 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 →	0 0 0 0 1 0 1 1 1 1 1 1 1 1 1
7	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0	0 0 0 0 1 0 1 1 1 1 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 →	0 0 0 0 0 1 0 1 1 1 1 1 1 1 1
8	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1	0 0 0 0 0 1 0 1 1 1 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	1 0 0 0 0 0 1 0 1 1 1 1 1 1 1
9	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	1 0 0 0 0 0 1 0 1 1 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 1 0 0 0 0 0 1 0 1 1 1 1 1 1
10	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	0 1 0 0 0 0 0 1 0 1 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 0 1 0 0 0 0 0 1 0 1 1 1 1 1
11	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	0 0 1 0 0 0 0 0 1 0 1 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 0 0 1 0 0 0 0 0 1 0 1 1 1 1
12	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	0 0 0 1 0 0 0 0 0 1 0 1 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 0 0 0 1 0 0 0 0 0 1 0 1 1 1
13	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	0 0 0 0 1 0 0 0 0 0 1 0 1 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 0 0 0 0 1 0 0 0 0 0 1 0 1 1
14	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	0 0 0 0 0 1 0 0 0 0 0 1 0 1 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 0 0 0 0 0 1 0 0 0 0 0 1 0 1
15	+X	1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0	0 0 0 0 0 0 1 0 0 0 0 0 1 0 1
		1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 →	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
16	−X	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 →	0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0

Fig. 10.28. Exemplu de înmulțire a două numere de 16 biți reprezentate în complement față de 2 (înmulțitorul negativ).

R1, R0 și Q, conform celei de-a doua metode a lui Robertson. În figura 10.27 se exemplifică o înmulțire fără corecție, în care înmulțitorul este pozitiv, iar în figura 10.28 o înmulțire cu corecție în care înmulțitorul este negativ.

Menționăm, de asemenea, că operația de înmulțire propriu-zisă a două numere reprezentate în complement față de 2 durează într-o unitate centrală microprogramată de tipul celei descrise aici 17 microcicli, fiind necesare trei microinstrucțiuni.

10.2.6.3. Microrutina de împărțire a două numere reprezentate în complement față de 2

Algoritmul prezentat în continuare se bazează pe cunoscuta metodă a împărțirii fără refacerea restului parțial [4]. Metoda este valabilă atît pentru orice poziție a virgulei, în cazul numerelor fracționare reprezentate în virgulă fixă, cît și, cu mici modificări, pentru numere întregi [5].

Fiind date două numere X, deîmpărțitul și Y, împărțitorul, operația de împărțire determină cîtul Q și restul R_n, astfel încît să fie satisfăcută relația următoare:

$$X = Q \cdot Y + R_n. \tag{10.17}$$

Metoda utilizează așa-numitele resturi parțiale. Restul parțial inițial este chiar deîmpărțitul:

$$r_0 = X. \tag{10.18}$$

Primul rest parțial este dat de relația:

$$r_1 = 2 \cdot r_0 + (1 - 2 \cdot q_1) \cdot Y, \tag{10.19}$$

unde q_1 este 1 sau 0 în funcție de semnul lui r_0 și Y după cum urmează:

$r_1 = 2 \cdot r_0 - Y$ pentru semne identice și

$r_1 = 2 \cdot r_0 + Y$ pentru semne diferite.

Următorul rest parțial este:

$$r_2 = 2 \cdot r_1 + (1 - 2 \cdot q_2) \cdot Y. \tag{10.20}$$

În general r_i este dat de relația:

$$r_i = 2 \cdot r_{i-1} + (1 - 2 \cdot q_i) \cdot Y. \tag{10.21}$$

Înmulțind în ambele părți cu 2^{-i} se obține:

$$2^{-i} \cdot r_i = 2^{-(i-1)} \cdot r_{i-1} + (2^{-i} - 2^{-(i-1)} \cdot q_i) \cdot Y. \tag{10.22}$$

Aplicînd repetat relația (10.22) pentru cazul în care cîtul este reprezentat pe 16 cifre binare rezultă:

$$2^{-15} \cdot r_{15} = X + \left(\sum_{i=1}^{15} 2^{-i} - \sum_{i=1}^{15} 2^{-(i-1)} \cdot q_i \right) \cdot Y. \tag{10.23}$$

Înlocuind în relația de mai sus suma progresiei geometrice:

$$\sum_{i=1}^{15} 2^{-i} = 1 - 2^{-15}$$

și împărțind cu Y în ambele părți se obține:

$$X/Y = \left(-1 + 2^{-15} + \sum_{i=1}^{15} 2^{-(i-1)} \cdot q_i\right) + 2^{-15} \cdot r_n/Y. \quad (10.24)$$

Comparînd cu relația (10.17) rezultă cîtul și restul împărțirii:

$$Q = -1 + 2^{-15} + \sum_{i=1}^{15} 2^{-(i-1)} \cdot q_i; \quad (10.25)$$

$$R_{15} = 2^{-15} \cdot r_{15}. \quad (10.26)$$

După cum se vede, cifrele cîtului q_i din relația (10.25) nu reprezintă cîtul corect. Pentru obținerea rezultatului corect trebuie adunat $-1 + 2^{-15}$. De asemenea, pentru obținerea restului corect trebuie efectuată o deplasare dreapta a restului parțial r_{15} cu 15 poziții.

Menționăm de asemenea că relația (10.25) care dă cîtul îl reprezintă ca număr fracționar impunînd deci condiția ca deîmpărțitul X să fie mai mic decît împărțitorul Y. Această condiție trebuie să fie asigurată înainte de începerea operației propriu-zise de împărțire.

În concluzie, în această metodă împărțitorul Y este adunat sau scăzut din restul parțial în funcție de semnele împărțitorului și restului parțial. Dacă aceste semne sînt identice se efectuează o scădere, cifra cîtului stabilindu-se „1". Dacă semnele sînt diferite se efectuează o adunare, cifra cîtului fiind în acest caz „0". În ambele situații se formează un nou rest parțial printr-o deplasare stînga. Procesul continuă pînă cînd restul devine egal cu zero sau pînă cînd se obține numărul dorit de cifre ale cîtului.

Algoritmul utilizat aici pentru împărțirea a două numere reprezentate în complement față de 2 are organigrama din figura 10.29. Algoritmul este valabil atît pentru operații în precizie simplă, cît și pentru operații în precizie dublă sau multiplă. Singura condiție care trebuie îndeplinită este, așa cum s-a spus mai sus, ca deîmpărțitul să fie mai mic decît împărțitorul.

În continuare vom descrie desfășurarea operației de împărțire în precizie dublă conform organigramei din figura 10.29.

Intrarea în microrutina de împărțire presupune că împărțitorul normalizat se află în registrul R0 și deîmpărțitul, de asemenea normalizat, în registrele R1, partea cea mai semnificativă, și R4, partea mai puțin semnificativă. Registrul R10 care va conține factorul de scalare trebuie să fie zero. La sfîrșitul împărțirii cîtul de 16 biți se va găsi în registrul Q iar restul va înlocui partea cea mai semnificativă a deîmpărțitului găsindu-se în registrul R1.

Algoritmul începe prin copierea împărțitorului și a părții mai semnificative a deîmpărțitului în registrele R3, respectiv R2, cu scopul comparării lor. În continuare se verifică dacă împărțitorul este diferit de zero. În cazul în care împărțitorul este zero operația de împărțire este abandonată. În caz

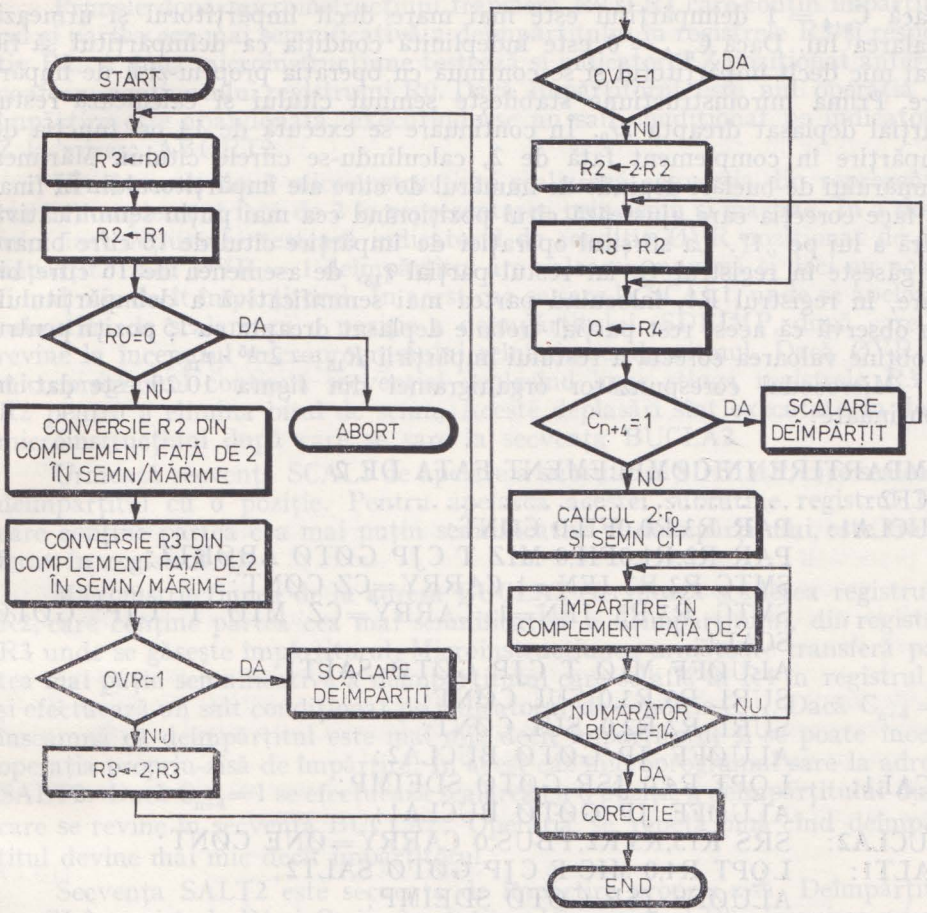


Fig. 10.29. Organigrama microrutinei de împărțire a două numere reprezentate în complement față de 2

contrar se efectuează conversia registrelor R3 și R2 din reprezentarea în complement față de 2 în reprezentarea prin mărime și semn. După conversia registrului R3, a deîmpărțitului deci, se verifică indicatorul de depășire ØVR. Dacă ØVR = 1 înseamnă că deîmpărțitul are valoarea maximă și deci nu poate fi mai mic decît împărțitorul. În acest caz urmează scalarea deîmpărțitului cu scopul de a-l face mai mic decît împărțitorul. Dacă ØVR = 0 împărțitorul este deplasat stînga pentru a elimina semnul. În continuare se testează indicatorul de condiție ØVR poziționat de data acesta de către împărțitor. Dacă ØVR=0 se elimină și semnul deîmpărțitului și se efectuează comparația între registrele R3 și R2. Această operație poziționează indicatorul C_{n+4}. După comparație, de fapt o scădere, se transferă cea mai puțin semnificativă parte a deîmpărțitului în registrul Q și se testează indicatorul de condiție C_{n+4}.

Dacă $C_{n+4} = 1$ deîmpărțitul este mai mare decît împărțitorul și urmează scalarea lui. Dacă $C_{n+4} = 0$ este îndeplinită condiția ca deîmpărțitul să fie mai mic decît împărțitorul și se continuă cu operația propriu-zisă de împărțire. Prima miroinstrucțiune stabilește semnul cîtului și calculează restul parțial deplasat dreapta $2r_0$. În continuare se execută de 14 ori funcția de împărțire în complement față de 2, calculîndu-se cifrele cîtului. Mărimea numărului de bucle depinde de numărul de cifre ale împărțitorului. În final se face corecția care ajustează cîtul poziționînd cea mai puțin semnificativă cifră a lui pe „1". La sfîrșitul operației de împărțire cîtul de 16 cifre binare se găsește în registrul Q, iar restul parțial r_{15}, de asemenea de 16 cifre binare, în registrul R1, înlocuind partea mai semnificativă a deîmpărțitului. Se observă că acest rest parțial trebuie deplasat dreapta cu 15 poziții pentru a obține valoarea corectă a restului împărțirii $R_{15} = 2^{-15} \cdot r_{15}$.

Microcodul corespunzător organigramei din figura 10.29 este dat în continuare:

```
;IMPARTIRE IN CØMPLEMENT FATA DE 2
DCF2: .....
BUCLA1:  PAR R3,R0,0FH,0 CØNT;
         PAR R2,R1,0FH,0 MIZ T CJP GØTØ ABØRT2;
         SMTC R2,R2 IEN=1 CARRY=CZ CØNT;
         SMTC R3,R3, IEN=1 CARRY=CZ MIØ T CJP GØTØ
         SCAL1;
         ALUØFF MIØ T CJP GØTØ SALT1;
         SURL R3,R3,0 SUL CØNT;
         SURL R2,R2,0 SUL CØNT;
         ALUØFF JP GØTØ BUCLA2;
SCAL1:   LQPT R4,0 JSR GØTØ SDEIMP;
         ALUØFF JP GØTØ BUCLA1;
BUCLA2:  SRS R15,R3,R2,YBUS,0 CARRY=ØNE CØNT;
SALT1:   LQPT R4,0 MIC F CJP GØTØ SALT2;
         ALUØFF JSR GØTØ SDEIMP;
         SDRL R2,R2,0 SDL CØNT;
         ALUØFF JP GØTØ BUCLA2;
SALT2:   ALUØFF GRD F LDCT CØUNT 00DH;
         DLN R1,R1,R0 IEN=1 RDU GRD T PUSH;
         TDIV R1,R1,R0 IEN=1 CARRY=CZ RDU CNT F RFCT;
         TDC R1,R1,R0 IEN=1 CARRY=CZ SUH CØNT;
         .....
SDEIMP:  PAR R1,R1,0FH,0 CØNT;
         ALUØFF MIS T CJP GØTØ NEG;
         PAR R1,R1,ADRQ,0 SDDL CØNT;
         ALUØFF JP GØTØ RET;
NEG:     PAR R1,R1,ADRQ,0 SDDH CØNT;
RET:     QMØV R4 CØNT;
         PAR R10,R10,0FH,0 CARRY=ØNE RTN;
ABØRT2:  .....
```

Primele două microinstrucțiuni transferă R0 și R1 care conțin împărțitorul și partea cea mai semnificativă a deîmpărțitului în registrele R3 și respectiv R2. A doua microinstrucțiune testează și indicatorul Z poziționat anterior conform conținutului registrului R0. Dacă împărțitorul este nul operația de împărțire este abandonată, executîndu-se un salt condiționat pe indicatorul Z la adresa ABØRT2.

Următoarele două microinstrucțiuni realizează conversia din reprezentarea în complement față de 2 în reprezentarea prin semn și mărime. În a doua microinstrucțiune se testează indicatorul de condiție ØVR poziționat de deîmpărțit. Dacă ØVR = 1 deîmpărțitul are valoarea maximă și deci nu poate fi mai mic decît împărțitorul. În acest caz se sare la SCAL1, unde se apelează subrutina de scalare cu o poziție a deîmpărțitului, SDEIMP, după care se revine la începutul microprogramului reluîndu-se algoritmul. Dacă ØVR=0 microprogramul continuă secvențial deplasînd logic stînga registrele R3 și R2 pentru a elimina bitul de semn. Aceste deplasări sînt executate în două microinstrucțiuni după care se sare la secvența BUCLA2.

Urmează secvența SCAL1 de apelare a subrutinei SDEIMP, care scalează deîmpărțitul cu o poziție. Pentru apelarea acestei subrutine registrul R4, care conține partea cea mai puțin semnificativă a deîmpărțitului, este transferat în registrul Q.

Microinstrucțiunea de la adresa BUCLA2 efectuează scăderea registrului R2, care conține partea cea mai semnificativă a deîmpărțitului, din registrul R3 unde se găsește împărțitorul. Microinstrucțiunea următoare transferă partea mai puțin semnificativă a deîmpărțitului care se află în R4 în registrul Q și efectuează un salt condiționat pe indicatorul de condiție C_{n+4}. Dacă $C_{n+4}=0$ înseamnă că deîmpărțitul este mai mic decît împărțitorul și se poate începe operația propriu-zisă de împărțire. În acest caz microprogramul sare la adresa SALT2. Dacă $C_{n+4}=1$ se efectuează scalarea cu o poziție a deîmpărțitului după care se revine în secvența BUCLA2. Operația se repetă pînă cînd deîmpărțitul devine mai mic decît împărțitorul.

Secvența SALT2 este secvența de împărțire propriu-zisă. Deîmpărțitul se află în registrele R1 și Q, iar împărțitorul în registrul R0. În prima microinstrucțiune se încarcă numărătorul de bucle. În a doua microinstrucțiune se efectuează o normalizare în precizie dublă obținîndu-se în R1 primul rest parțial, deîmpărțitul, deplasat stînga cu o poziție. De asemenea, în această microinstrucțiune semnul cîtului, egal cu semnul deîmpărțitului, este încărcat în poziția cea mai puțin semnificativă a registrului Q deplasat stînga. Tot în această microinstrucțiune se încarcă în stivă adresa următoare care este de fapt adresa de început a buclei de împărțire. În continuare se execută de 14 ori instrucțiunea de împărțire în complement față de 2, calculîndu-se în registrul R1 resturile parțiale și în registrul Q, cifrele cîtului. În final o microinstrucțiune separată efectuează corecția cîtului poziționînd ultima sa cifră binară pe „1". La sfîrșitul acestei secvențe cîtul se află în registrul Q și restul parțial r_{15}, în registrul R1. Menționăm că, în continuare, pentru obținerea corectă a rezultatelor trebuie ținut cont, pe de o parte, de factorul de scară, iar pe de altă parte de faptul că restul parțial r_{15} trebuie deplasat dreapta cu 15 poziții.

Subrutina de scalare a deîmpărțitului, SDEIMP, începe printr-o microinstrucțiune de poziționare a indicatorului de condiție S, semnul. Microinstrucțiunea următoare testează acest indicator executînd un salt condiționat la adresa NEG. Dacă deîmpărțitul este pozitiv se va deplasa aritmetic dreapta cu o poziție introducîndu-se în stînga o cifră binară „0". Dacă deîmpărțitul este negativ microinstrucțiunea de la adresa NEG efectuează de asemenea o deplasare aritmetică dreapta cu deosebirea că cifra introdusă în stînga este acum „1". În final ultimele două microinstrucțiuni transferă registrul Q, ce conține partea cea mai puțin semnificativă a deîmpărțitului, înapoi în R4, incrementează registrul R10 care conține factorul de scară și execută revenirea din subrutină.

10.3. UN PROCESOR DE SEMNAL MICROPROGRAMAT CU ARITMETICĂ DISTRIBUITĂ

10.3.1. ARITMETICĂ DISTRIBUITĂ

Un alt domeniu de aplicare tipică a microprocesoarelor *bit-slice* este acela al procesării de semnal. Gama unor astfel de aplicații variază în complexitate de la procesoare mici, specializate, pentru implementarea filtrelor numerice, pînă la mașini mari, programabile, orientate pe transformate Fourier rapide (Fast Fourier Transform — FFT).

În acest paragraf vom prezenta un procesor de semnal microprogramat cu performanțe medii destinat în primul rînd implementării funcțiilor de filtrare numerică, dar și, prin extinderea memoriei de control, implementării de funcții mai complicate cum sînt FFT [6].

Este cunoscut că în majoritatea aplicațiilor procesoarele de semnal trebuie să calculeze o sumă algebrică de produse de forma:

$$y = \sum_{j=1}^{L} a_j \cdot x_j = a^T \cdot x. \tag{10.27}$$

Relația (10.27) reprezintă produsul a doi vectori. În această relație a_j sînt de obicei coeficienți constanți și x_j variabile.

Dacă $|x_j| < 1$ și se utilizează o reprezentare a numerelor în virgulă fixă în complement față de 2, CF2, pe B biți, atunci x_j se poate scrie [4]:

$$x_j = (x_{j0}, x_{j1}, \ldots, x_{jB-1})_{CF2} = -x_{j0} + \sum_{i=1}^{B-1} x_{ji} \cdot 2^{-i}, \tag{10.28}$$

unde $x_{ji} = 0$ sau 1 și x_{j0} reprezintă cifra de semn.

Înlocuind în (10.27) se obține:

$$y = \sum_{j=1}^{L} \sum_{i=1}^{B-1} a_j \cdot x_{ji} \cdot 2^{-i} - \sum_{j=1}^{L} a_j \cdot x_{j0} \quad \text{sau}$$

$$y = \sum_{i=1}^{B-1} 2^{-i} \sum_{j=1}^{L} a_j \cdot x_{ji} - \sum_{j=1}^{L} a_j \cdot x_{j0}. \tag{10.29}$$

Definind:

$$F(x_{1i}, x_{2i}, ..., x_{Li}) = \sum_{j=1}^{L} a_j \cdot x_{ji} \tag{10.30}$$

suma y devine:

$$y = \sum_{i=1}^{B-1} 2^{-i} \cdot F(x_{1i}, x_{2i}, ..., x_{Li}) - F(x_{10}, x_{20}, ..., x_{L0}). \tag{10.31}$$

Ecuația (10.30) reprezintă implementarea produsului a doi vectori prin utilizarea aritmeticii distribuite. Într-adevăr funcția F poate fi implementată cu ajutorul unei memorii de tip PROM cu L biți de adresă și un număr de ieșiri determinat de precizia cerută de problemă. De obicei, acest număr de biți este $B = 16$ pentru operații în precizie simplă.

Ca exemplu, să presupunem $B = 16$ și $L = 8$. Suma y va fi conform formulelor (10.27) și (10.30):

$$y = \sum_{j=1}^{8} a_j \cdot x_j = \sum_{i=1}^{15} 2^{-i} \cdot F(x_{1i}, x_{2i}, ..., x_{8i}) - F(x_{10}, x_{20}, ..., x_{80}), \tag{10.32}$$

unde

$$F(x_{1i}, x_{2i}, ..., x_{8i}) = \sum_{j=1}^{8} a_j \cdot x_{ji}.$$

Ecuația (10.31) se poate implementa hardware cu schema din figura 10.30. După inițializarea acumulatorului cele opt registre de deplasare se vor încărca cu variabilele x_j, $j = 1 \ldots 8$. În continuare prin 15 comenzi de adunare și deplasare se produce în acumulator un rezultat parțial din care în final se scade

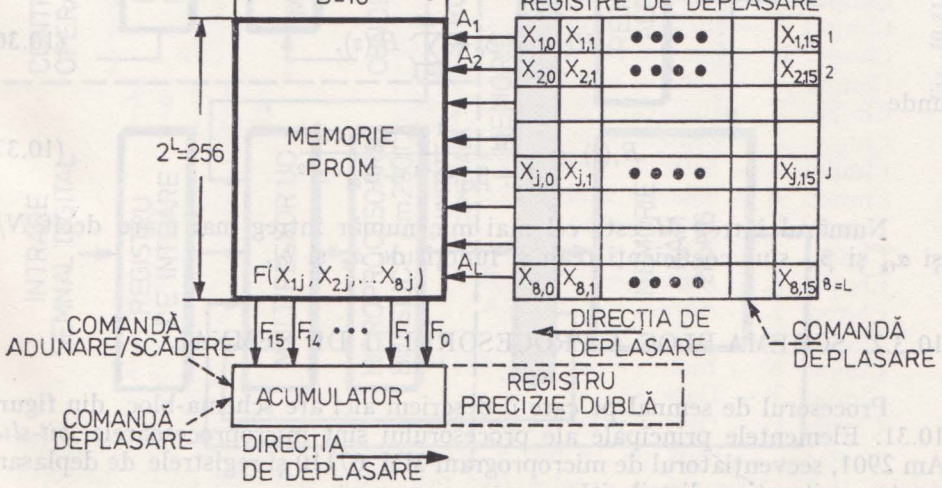


Fig. 10.30. Implementarea produsului a doi vectori prin aritmetică distribuită.

$F(x_{10}, x_{20}, ..., x_{80})$ obținîndu-se suma dată de (10.31). Se observă că adunarea a opt produse se poate realiza prin aritmetică distribuită în 16 cicli, timp echivalent cu timpul necesar doar pentru înmulțirea prin microcod a două numere (v. § 10.2.6.2).

În aplicațiile de filtrare numerică procesorul de semnal calculează sume de tipul (10.30) pentru a obține un set de ecuații cu diferențe finite, ecuații derivate din funcția de transfer a filtrului digital [7]. După cum se știe, această funcție de transfer rezultă din aplicarea transformatei Z unei ecuații cu diferențe finite care caracterizează funcționarea filtrului numeric. Funcția de transfer $H(z)$ este de tipul:

$$H(z) = \frac{\sum_{k=0}^{N} a_k \cdot z^{-k}}{1 + \sum_{k=1}^{N} b_k \cdot z^{-k}}, \tag{10.33}$$

unde a_k și b_k sînt constante reale care specifică răspunsul în frecvență al filtrului. Dacă $N > 2$ funcția de transfer se poate realiza prin conectarea în cascadă sau în paralel a unor module de ordinul doi. Dacă se folosește o conectare în cascadă, atunci funcția de transfer este de forma:

$$H(z) = \prod_{i=1}^{M} A_i(z), \tag{10.34}$$

unde

$$A_i(z) = \frac{\alpha_{i0} + \alpha_{i1} \cdot z^{-1} + \alpha_{i2} \cdot z^{-2}}{1 + \alpha_{i3} \cdot z^{-1} + \alpha_{i4} \cdot z^{-2}}. \tag{10.35}$$

Dacă se folosește o conectare în paralel atunci:

$$H(z) = \beta_0 + \sum_{i=1}^{M} B_i(z), \tag{10.36}$$

unde

$$B_i(z) = \frac{\beta_{i1} \cdot z^{-1} + \beta_{i2} \cdot z^{-2}}{1 + \beta_{i3} \cdot z^{-1} + \beta_{i4} \cdot z^{-2}}. \tag{10.37}$$

Numărul întreg M este cel mai mic număr întreg mai mare decît $N/2$ și α_{ik} și β_{ik} sînt coeficienți reali și funcții de a_k și b_k.

10.3.2. SCHEMA-BLOC A PROCESORULUI DE SEMNAL

Procesorul de semnal pe care îl descriem aici are schema-bloc din figura 10.31. Elementele principale ale procesorului sînt microprocesoarele *bit-slice* Am 2901, secvențiatorul de microprogram MM 67110 și registrele de deplasare pentru aritmetica distribuită.

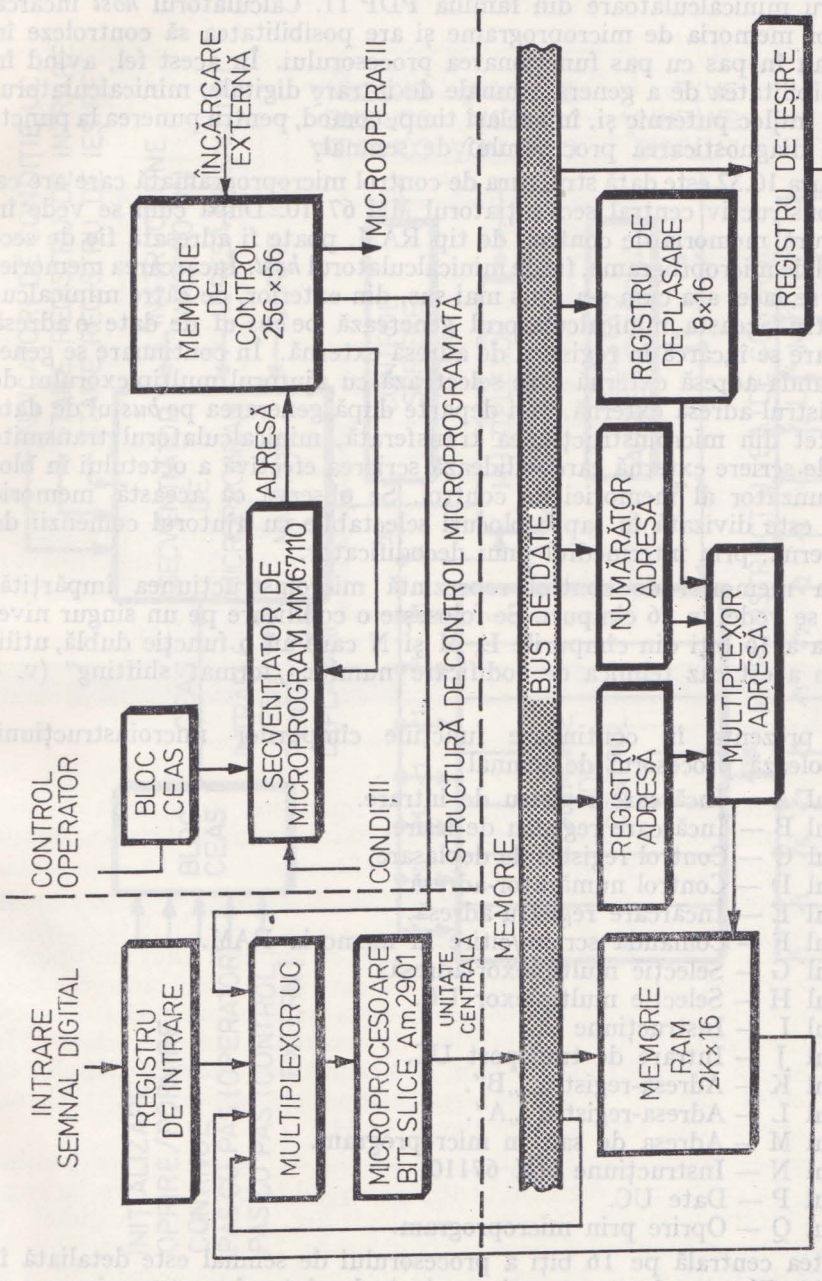


Fig. 10.31. Schema-bloc a unui procesor de semnal

Procesorul de semnal din figura 10.31 este conceput ca un dispozitiv periferic pentru minicalculatoare din familia PDP 11. Calculatorul *host* încarcă din exterior memoria de microprograme și are posibilitatea să controleze în automat sau în pas cu pas funcționarea procesorului. În acest fel, avînd în vedere posibilitatea de a genera semnale de intrare digitale, minicalculatorul oferă și un mijloc puternic și, în același timp, comod, pentru punerea la punct, testarea și diagnosticarea procesorului de semnal.

În figura 10.32 este dată structura de control microprogramată care are ca element constructiv central secvențiatorul MM 67110. După cum se vede în această figură, memoria de control, de tip RAM, poate fi adresată fie de secvențiatorul de microprograme, fie de minicalculatorul *host*. Încărcarea memoriei de control se face, așa cum s-a spus mai sus, din exterior, de către minicalculator. Pentru aceasta minicalculatorul generează pe *bus*-ul de date o adresă de 8 biți care se încarcă în registrul de adresă externă. În continuare se generează comanda-adresă externă care selectează cu ajutorul multiplexorului de adresă registrul-adresă externă. Mai departe după generarea pe *bus*-ul de date a unui octet din microinstrucțiunea transferată, minicalculatorul transmite comanda de scriere externă care validează scrierea efectivă a octetului în blocul corespunzător al memoriei de control. Se observă că această memorie de control este divizată în șapte blocuri selectabile cu ajutorul comenzii de scriere externă, prin intermediul unui decodificator.

Ieșirea memoriei de control reprezintă microinstrucțiunea împărțită, după cum se vede, în 16 cîmpuri. Se folosește o codificare pe un singur nivel cu excepția a 16 biți din cîmpurile L, M și N care au o funcție dublă, utilizîndu-se în acest caz tehnica de codificare numită „format shifting" (v. § 7.2.2.2).

Vom prezenta în continuare funcțiile cîmpurilor microinstrucțiunii care controlează procesorul de semnal:

Cîmpul A — Încărcare registru de intrare.
Cîmpul B — Încărcare registru de ieșire.
Cîmpul C — Control registre de deplasare
Cîmpul D — Control numărător-adresă.
Cîmpul E — Încărcare registru-adresă.
Cîmpul F — Comandă scriere/citire în memoria RAM.
Cîmpul G — Selecție multiplexor-adresă.
Cîmpul H — Selecție multiplexor UC.
Cîmpul I — Instrucțiune UC.
Cîmpul J — Intrare de transport UC.
Cîmpul K — Adresa-registru „B".
Cîmpul L — Adresa-registru „A".
Cîmpul M — Adresa de salt în microprogram.
Cîmpul N — Instrucțiune MM 67110.
Cîmpul P — Date UC.
Cîmpul Q — Oprire prin microprogram.

Unitatea centrală pe 16 biți a procesorului de semnal este detaliată în figura 10.33. Elementele constructive principale sînt cele patru microprocesoare Am 2901. Se observă multiplicatorul opțional care poate fi utilizat pen-

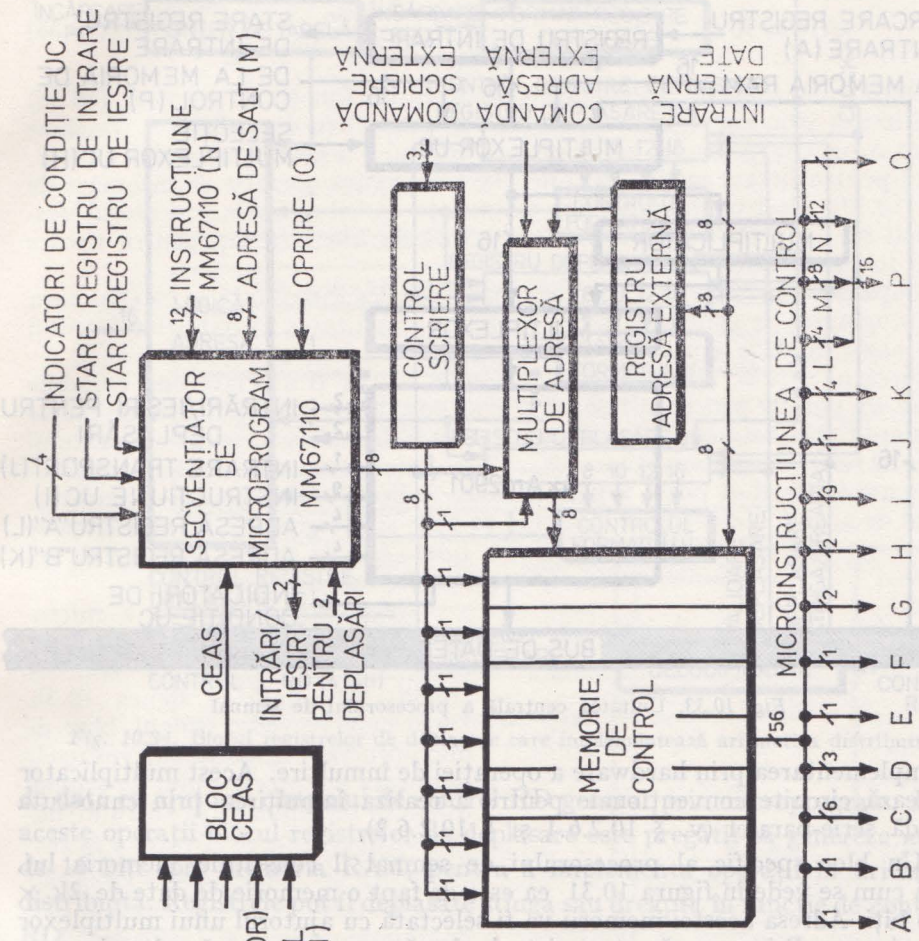


Fig. 10.32. Structura de control microprogramată a procesorului de semnal

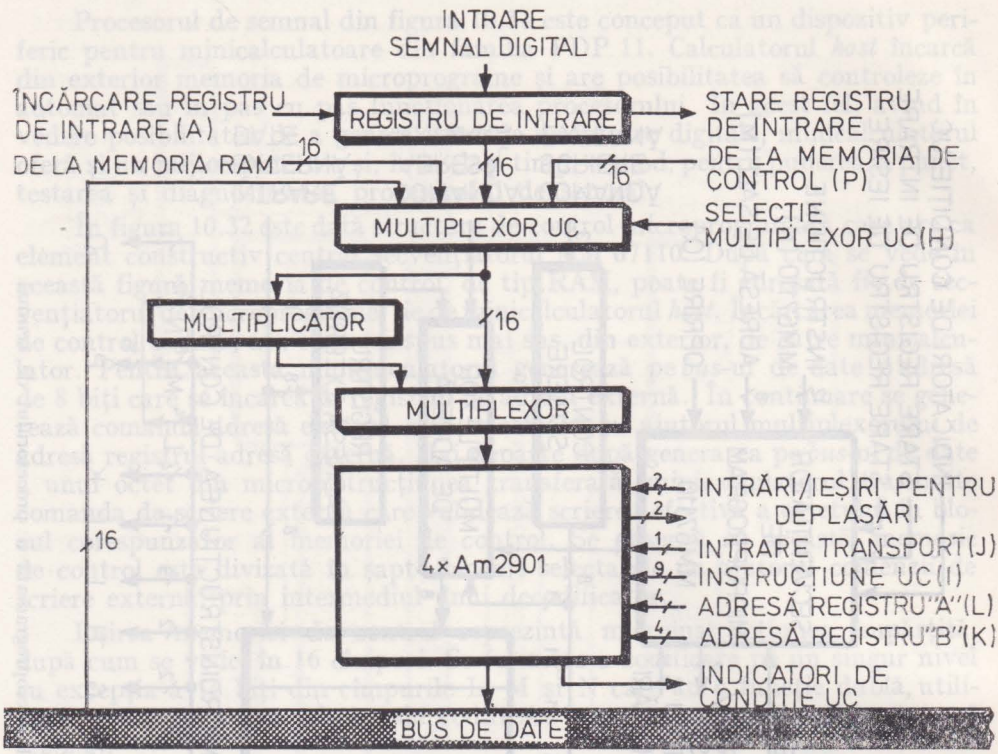


Fig. 10.33. Unitatea centrală a procesorului de semnal

tru implementarea prin hardware a operației de înmulțire. Acest multiplicator utilizează circuite convenționale pentru a realiza înmulțirea prin cunoscuta metodă serie-paralel (v. § 10.2.6.1 și § 10.2.6.2).

Un bloc specific al procesorului de semnal îl constituie memoria lui. După cum se vede în figura 10.31 ea este de fapt o memorie de date de 2k × × 16 biți. Adresa acestei memorii va fi selectată cu ajutorul unui multiplexor din trei surse. Prima sursă este registrul-adresă care memorează adresele constantelor sau variabilelor mai des folosite. A doua sursă e numărătorul-adresă utilizat pentru indexare rapidă în tabelele de constante sau de variabile. În fine, a treia sursă, specifică procesorului de semnal, este blocul registrelor de deplasare folosit pentru implementarea aritmeticii distribuite. Schema detaliată a acestui bloc este dată în figura 10.34.

Pointerul de tabelă este inițializat pe adresa de memorie care conține $F(0, 0, ..., 0)$. În continuare registrul pentru controlul formatului este încărcat cu un cuvînt care formează registrele de deplasare, precizînd lungimea B a fiecărui registru și numărul L de registre utilizate. Lungimea registrelor de deplasare poate fi de 8, 10, 12 sau 16 biți, iar numărul de registre $L \leqslant 8$. Variabilele x_i, $i = 1 \div L$, sînt încărcate în registrele de deplasare de pe *bus*-ul

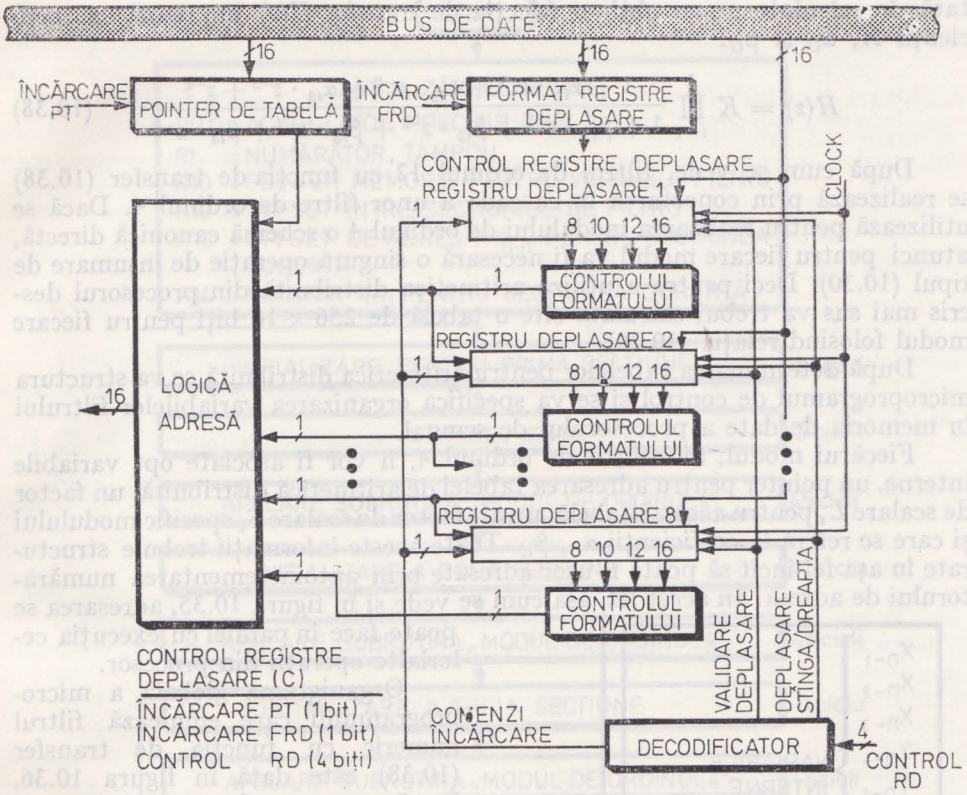


Fig. 10.34. Blocul registrelor de deplasare care implementează aritmetica distribuită

de date cu ajutorul cîmpului de control RD generat prin microprogram. După aceste operații blocul registrelor de deplasare este pregătit să genereze adrese de 16 biți spre memoria RAM, pentru a implementa operații în aritmetică distribuită. Registrele pot fi deplasate stînga sau dreapta în funcție de controlul RD.

10.3.3. IMPLEMENTAREA UNUI FILTRU NUMERIC CU AJUTORUL UNUI PROCESOR DE SEMNAL

Filtrul ales în [6] ca exemplu pentru a fi implementat cu ajutorul procesorului de semnal descris mai sus este un filtru eliptic trece-bandă cu benzi de trecere de 500—600 Hz și 1 000—1 100 Hz, abaterea în banda de trecere de 0,2 dB, atenuarea în banda de oprire de 56 dB și frecvența de eșantionare de 8 kHz. Plecînd de la aceste date, cu ajutorul unui sistem de proiectare asis-

tată de calculator, s-au obținut funcția de transfer $H(z)$ împreună cu coeficienții K, α_{ij} și β_{ij}:

$$H(z) = K \prod_{i=1}^{3} \frac{1 + \alpha_{i1} \cdot z^{-1} + \alpha_{i2} \cdot z^{-2} + \alpha_{i3} \cdot z^{-3} + z^{-4}}{1 + \beta_{i1} \cdot z^{-1} + \beta_{i2} \cdot z^{-2} + \beta_{i3} \cdot z^{-3} + \beta_{i4} \cdot z^{-4}}. \quad (10.38)$$

După cum se vede, filtrul de ordinul 12 cu funcția de transfer (10.38) se realizează prin conectarea în cascadă a unor filtre de ordinul 4. Dacă se utilizează pentru realizarea modulului de ordinul 4 o schemă canonică directă, atunci pentru fiecare modul va fi necesară o singură operație de însumare de tipul (10.30). Deci pentru a utiliza aritmetica distribuită din procesorul descris mai sus va trebui calculată cîte o tabelă de 256 × 16 biți pentru fiecare modul folosind relația (10.29).

După determinarea tabelelor pentru aritmetica distribuită se va structura microprogramul de control și se va specifica organizarea variabilelor filtrului în memoria de date a procesorului de semnal.

Fiecărui modul, aici filtru de ordinul 4, îi vor fi asociate opt variabile interne, un pointer pentru adresarea tabelei de aritmetică distribuită, un factor de scalare L_i pentru această tabelă, un alt factor de scalare S_i specific modulului și care se referă la coeficienții α_{ij}, β_{ij}. Toate aceste informații trebuie structurate în așa fel încît să poată fi ușor adresate prin autoincrementarea numărătorului de adresă. În acest fel, așa cum se vede și în figura 10.35, adresarea se poate face în paralel cu execuția celorlalte operații din procesor.

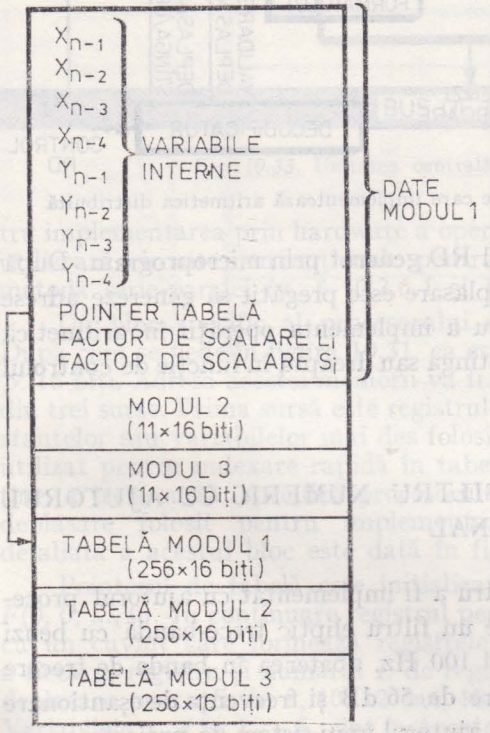


Fig. 10.35. Organizarea variabilelor filtrului

Organigrama globală a microprogramului care emulează filtrul numeric cu funcția de transfer (10.38) este dată în figura 10.36. După cum se vede, primul bloc de microprogram inițializează resursele hardware ale procesorului de semnal utilizate de emulatorul filtrului numeric. În continuare, pentru a se realiza filtrul de ordinul 12 se apelează de trei ori subrutina de calcul al modulului de ordinul 4. În figura 10.36 este trecut și numărul de microcicli necesar execuției fiecărei părți din organigramă. În cazul unui ciclu de microinstrucțiune de 300 ns rezultă frecvența maximă de eșantionare a filtrului numeric:

$$f_{max} = \frac{1}{141 \times 300 \text{ ns}} = 23{,}6 \text{ kHz}.$$

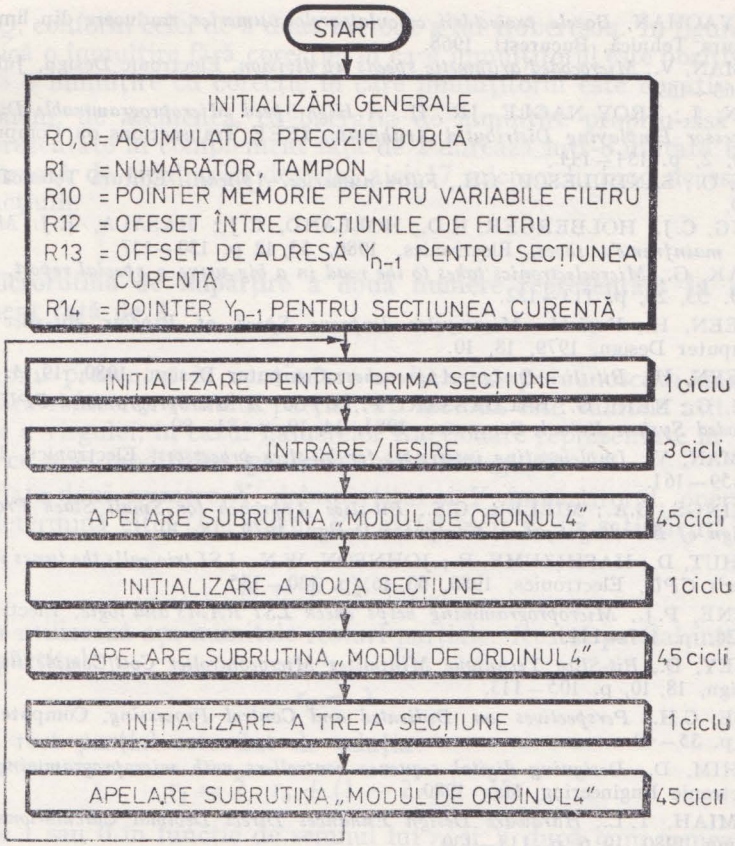


Fig. 10.36. Structura globală a microprogramului care emulează filtrul numeric cu funcția de transfer (10.38)

Menționăm că memorarea variabilelor în memoria de date, deși oferă o flexibilitate deosebită, în special în cazul implementării filtrelor de ordin mai mare, necesită un timp suplimentar atît pentru încărcarea lor în registrele de deplasare, cît și pentru realizarea operațiilor de întîrziere. Pentru filtre rapide de ordin mic aceste variabile pot fi păstrate în registrele de deplasare mărindu-se în acest fel rata de eșantionare la 155, 9 kHz [6].

BIBLIOGRAFIE

1. LUPU, C.; FIERĂSTRĂU, D.; VERZEA, GH., *Canal selector microprogramat pentru discuri magnetice cu capete fixe*, comunicare la Sesiunea de comunicări științifice jubiliară — Un deceniu de activitate a I.T.C., București, Noiembrie, 1978.
2. * * * *Build a Microcomputer*, Chapter IV, The Data Path-Part II, Advanced Micro Devices Inc., Sunnyvale, California, 1979.
3. BALTAC, V., colectiv, *FELIX C-256. Structura și programarea calculatorului*, Editura Tehnică, București, 1974.

4. CHU, YAOHAN, *Bazele proiectării calculatoarelor numerice*, traducere din limba engleză, Editura Tehnică, București, 1968.
5. COLEMAN, V., *Microcoded arithmetic speeds up division*, Electronic Design, July 19, 1980, p. 163—169.
6. ZEMAN, J.; TROY NAGLE, JR., H., *A High-Speed Microprogrammable Digital Signal Processor Employing Distributed Arithmetic*, IEEE Transactions on Computers, 1980, C-29, 2, p. 134—144.
7. RADU, O.; SĂNDULESCU, GH., *Filtre numerice. Aplicatii*, Editura Tehnică, București, 1979.
8. ALSING, C.J.; HOLBERGER, K.D.; HOLLAND, C.J.; RASALA, E.J., *Minicomputer fills mainframe's shoes*, Electronics, 1980, 53, 12 p. 130—137.
9. BASSAK, G., *Microelectronics takes to the road in a big way: a special report*, Electronics, 1980, 53, 25, p. 113—122.
10. BRINEEN, H., *Variable Microcycles Improve Speed of Bipolar Bit-slice Processors*, Computer Design, 1979, 18, 10.
11. BRINEEN, H., *Bit-slice Design Approaches*, Computer Design, 1980, 19, 4.
12. BUCCI, G.; NERI, G.; BALDASSARI, F., *MP80: A Microprogrammed CPU with a Microcoded System Kernel*, Computer, 1981, 14, 10, p. 81—90.
13. COLEMAN, V., *Implementing interrupts for bit-slice processors*, Electronics, 1980, 53, 10, p. 159—161.
14. CUMMINGS, G.A.; MILLER, G.S., *Bit-slice Approach for Small Stack Processor Aids Design of Blending System*, Computer Design, 1979, 18, 10.
15. DICKHUT, D.; HASHIZUME, B.; JOHNSON, W.N., *LSI trio calls the tunes in microcomputer's CPU*, Electronics, 1980, 53, 16, p. 130—135.
16. EUGENE, P.J., *Microprogramming helps check LSI RAMs and logic*, Electronics, 1980, 53, 26, p. 137—141.
17. HOOLEY, D., *Bit-Slice Technique Minimizes Microcontroller Cost/Complexity*, Computer Design, 18, 10, p. 105—113.
18. HOUSE, C.H., *Perspectives on Dedicated and Control Processing*, Computer, 1980, 13, 12, p. 35—49.
19. IBRAHIM, D., *Designing digital sequence controllers with microprogramming techniques*, Electronic Engineering, May 1980.
20. JEREMIAH, T.L., *Hardware Design Enhances Direct Decimal Calculations*, Computer Design, 1980, 19, 6, p. 118—130.
21. KARWOSKI, R.J., *Implement digital filters efficiently with a specially configured computer*, Electronic Design, 1979, 27, 18, p. 110—116.
22. KARWOSKI, R. J., *Computer implemented digital bandpass filters give almost any output response*, Electronic Design, 1979, 27, 26, p. 50—55.
23. LALA, P.K.; CROOKS, R.W., *Burst error correction in disk drives using PROMs*, Electronic Engineering, 1980, 52, 634, p. 60—67.
24. MESSIAS, H.; MCKEEVER, B., *Team an array processor with a μP for super performance in data comm*, Electronic Design, 1980, 28, 8, p. 115—120.
25. REDINBO, G.R., *Finite Field Arithmetic on an Array Processor*, IEEE Transactions on Computers, 1979, C-28, 7, p. 461—471.
26. REUSS, J.L., *PROM replaces individual gates in pseudorandom sequence generator*, Electronic Design, 1979, 27, 15, p. 122—124.
27. * * * *Un μP 10800 en ECL dans le mini-ordinateur Mitra 525*, Electronique et applications industrielles, 1980, 286/1-6-80, p. 3—4.
28. SWARTZLANDER, E.E., *Microprogrammed Control of Specialized Processors*, IEEE Transactions on Computers, 1979, C-28, 12, p. 930—933.
29. WILEY, P., *Interfacing Peripherals Directly to an Array Processor*, Computer Design, 1979, 18, 8, p. 158—164.
30. WOLFE, C.F., *Bit-slice processors come to mainframe design*, Electronics, 1980, 53, 5, p. 118—123.